Parul® University

Unit 6-
Transaction Processing

Subject Code: 303105203

Prof. S.W.Thakare
Assistant Professor,
Computer science & Engineering

Parul®
University

DIGITAL LEARNING CONTENT

CHAPTER-6

Transaction Processing

Transaction

- Collection of operations that form a single logical unit of work are called transactions.
- A transaction is a unit of program execution that accesses and possibly updates various data items.
- A transaction is a logical unit of work that contains one or more SQL statements.
- Usually, a transaction is initiated by a user program written in high-level data-manipulation language or programming language, where it is delimited by statements of the form **begin transaction** and **end transaction**.

ACID property

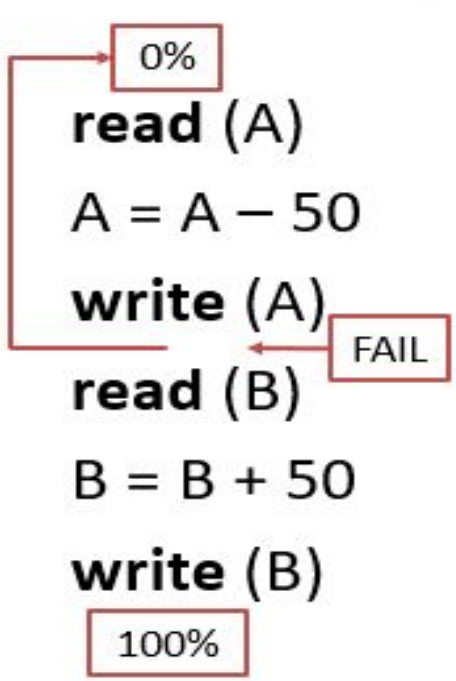
- To ensure integrity of the data, database system must maintain the following properties:
  - Atomicity
  - Consistency
  - Isolation
  - Durability

ACID property

- Atomicity: Either all operations of the transaction are reflected properly in the database, or none are.
- Consistency: Execution of a transaction in isolation i.ie., with no other transaction executing concurrently, preserves the consistency of the database.
- Isolation: Even though multiple transactions may execute concurrently, the system guarantees that, for every pair of transactions Ti and Tj, it appears to Ti that either Tj finished execution before Ti started or Tj started execution after Ti finished. Thus, each transaction is unaware of other transactions executing concurrently in the system.
- Durability: After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

Atomicity

- This property states that a transaction must be treated as an atomic unit, i.e., either all of its operations are executed or none.
- For example, consider a transaction to transfer Rs. 50 from account A to account B.
- In this transaction, if Rs. 50 is deducted from account A then it must be added to account B.



Consistency

- The database must remain in a consistent state after any transaction.
- If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.
- In our example, total of A and B must remain same before and after the execution of transaction.

A=500, B=500

A+B=1000

read (A)

A = A − 50

write (A)

read (B)

B = B + 50

write (B)

A=450, B=550

A+B=1000

Isolation

- Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions.
- Intermediate transaction results must be hidden from other concurrently executed transactions.
- In the example once your transaction start from step one its result should not be accessed by any other transaction until last step (step 6) is completed.

A=500, B=500

read (A)	1
A = A – 50	2
write (A)	3
read (B)	4
B = B + 50	5
write (B)	6

A=450, B=550

Durability

- After a transaction completes successfully, the changes it has made to the database persist(permanent), even if there are system failures.
- Once our transaction is completed up to last step (step 6) its result must be stored permanently. It should not be removed if the system fails.

A=500, B=500

read (A)	1
A = A – 50	2
write (A)	3
read (B)	4
B = B + 50	5
write (B)	6

A=450, B=550

Transaction State

- A transaction may not always complete its execution successfully, such transaction is termed **aborted**.
- Once the changes caused by an aborted transaction have been undone, we say that the transaction has been **rolled back**.
- A transaction that completes its execution successfully is said to be **committed**.
- Once the transactions are committed, we cannot undo its effects by aborting it. The only way to undo the effects of a committed transaction is to execute a **compensating transaction**.

Transaction State

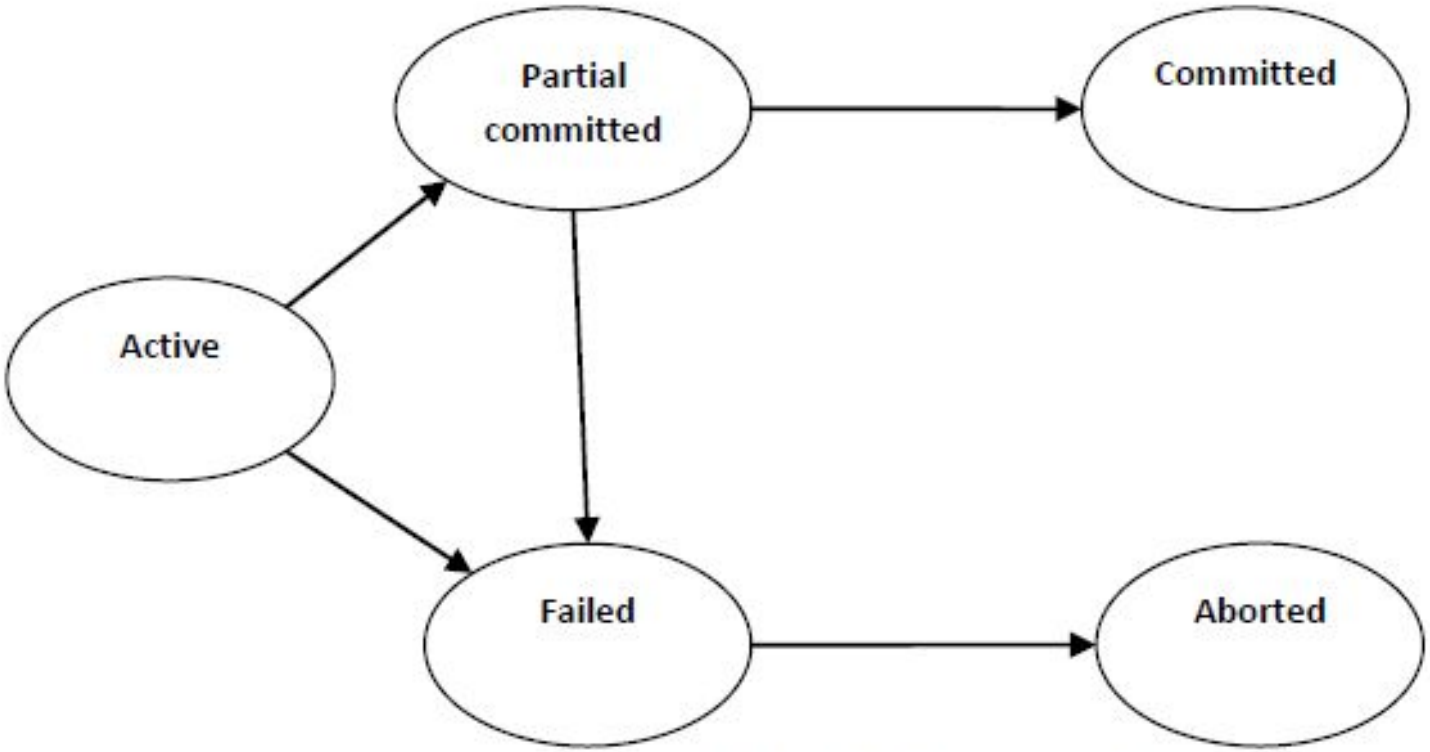


Fig. State Transition Diagram

Image source : Google

Transaction State

 A transaction must be in one of the following states:

- **Active**, the initial state; the transaction stays in this state while it is executing.
- **Partially committed**, after the final statement has been executed.
- **Failed**, after the discovery that normal execution can no longer proceed.
- **Aborted**, after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.
- **Committed**, after successful transaction.

The Transaction Log

- Keeps track of all transactions that update the database. It contains:
  - A record for the beginning of transaction
  - For each transaction component (SQL statement)
    - Type of operation being performed (update, delete, insert)
    - Names of objects affected by the transaction (the name of the table)
    - “Before” and “after” values for updated fields
    - Pointers to previous and next transaction log entries for the same transaction
  - The ending (COMMIT) of the transaction
- This increases processing overhead but the advantage is the **ability to restore a corrupted database**

Implementation of Atomicity and Durability

- The **recovery-management** component of a database system implements the support for atomicity and durability.
- E.g. the ***shadow-database*** scheme:
  - all updates are made on a *shadow copy* of the database
    - **db_pointer** is made to point to the updated shadow copy after
      - the transaction reaches partial commit and
      - all updated pages have been flushed to disk.

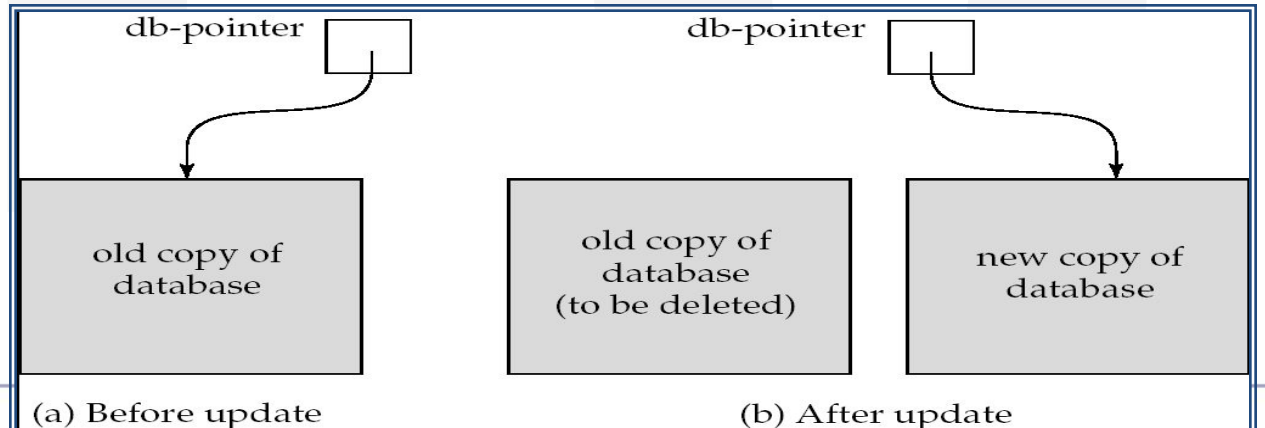


Image source : Google

Implementation of Atomicity and Durability

- db_pointer always points to the current consistent copy of the database.
  - In case transaction fails, old consistent copy pointed to by **db_pointer** can be used, and the shadow copy can be deleted.
- The shadow-database scheme:
  - Assumes that only one transaction is active at a time.
  - Assumes disks do not fail
  - Useful for text editors, but
    - extremely inefficient for large databases (why?)
      - Variant called shadow paging reduces copying of data, but is still not practical for large databases
  - Does not handle concurrent transactions

Concurrent Executions

- Multiple transactions are allowed to run concurrently in the system. Advantages are:
  - **increased processor and disk utilization**, leading to better transaction *throughput*
    - E.g. one transaction can be using the CPU while another is reading from or writing to the disk
  - **reduced average response time** for transactions: short transactions need not wait behind long ones.
- **Concurrency control schemes** – mechanisms to achieve isolation
  - that is, to control the interaction among the concurrent transactions in order to prevent them from destroying the consistency of the database

Schedules

- **Schedule** – a sequences of instructions that specify the chronological order in which instructions of concurrent transactions are executed
  - a schedule for a set of transactions must consist of all instructions of those transactions
  - must preserve the order in which the instructions appear in each individual transaction.
- A transaction that successfully completes its execution will have a commit instructions as the last statement
  - by default transaction assumed to execute commit instruction as its last step
- A transaction that fails to successfully complete its execution will have an abort instruction as the last statement

Schedule - 1

- Let T_1 transfer $50 from A to B, and T_2 transfer 10% of the balance from A to B.
- A serial schedule in which T_1 is followed by T_2 :

$T1$	$T2$
read(A)	
$A := A - 50$	
write (A)	
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)

Image source : Korth

Schedule - 2

- A serial schedule where T_2 is followed by T_1

T_1	T_2
	read(A) $temp := A * 0.1$ $A := A - temp$ write(A) read(B) $B := B + temp$ write(B)
read(A) $A := A - 50$ write(A) read(B) $B := B + 50$ write(B)	

Image source : Korth

Schedule - 3

- Let T_1 and T_2 be the transactions defined previously.
- The following schedule is not a serial schedule, but it is equivalent to Schedule 1.

T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
read(B)	
$B := B + 50$	
write(B)	
	read(B)
	$B := B + temp$
	write(B)

Schedule - 4

The following concurrent schedule does not preserve the value of *(A + B)*.

T_1	T_2
read(A)	
$A := A - 50$	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
write(A)	
read(B)	
$B := B + 50$	
write(B)	
	$B := B + temp$
	write(B)

Scheduling

- Scheduling is the technique of preserving the order of the operations from one transaction to another while executing such concurrent transactions.
- A series of operations from one transaction to another transaction is known as a schedule.

Why is it required?

- Transactions are a set of instructions that perform operations on databases. When multiple transactions are running concurrently, then a sequence is needed in which the operations are to be performed because at a time, only one operation can be performed on the database.
- This sequence of operations is known as Schedule, and this process is known as Scheduling.

Image source : Korth

Scheduling

- When multiple transactions execute simultaneously in an unmanageable manner, then it might lead to several problems, which are known as concurrency problems. In order to overcome these problems, scheduling is required.

Types of Schedules

There are mainly two types of scheduling -

1. Serial Schedule
2. Non-serial Schedule Further, they are divided

Scheduling

Serial Schedule

- As the name says, all the transactions are executed serially one after the other.
- In serial Schedule, a transaction does not start execution until the currently running transaction finishes execution.
- This type of execution of the transaction is also known as non-interleaved execution.
- Serial Schedule are always recoverable, cascades, strict and consistent. A serial schedule always gives the correct result.
- Consider two transactions T1 and T2 shown above, which perform some operations. If it has no interleaving of operations, then there are the following two possible outcomes –
- Either execute all T1 operations, which were followed by all T2 operations.

Image source : Korth

Scheduling

- The Schedule shows the serial Schedule where T1 is followed by T2, i.e. T1 -> T2. Where R(A) -> reading some data item ‘A’. And, W(B) -> writing/updating some data item ‘B’.
- If n = number of transactions, then a number of serial schedules possible = n!.
- Therefore, for the above 2 transactions, a total number of serial schedules possible = 2.

Image source : Korth

Scheduling

Non-serial Schedule

- In a non-serial Schedule, multiple transactions execute concurrently/simultaneously, unlike the serial Schedule, where one transaction must wait for another to complete all its operations.
- In the Non-Serial Schedule, the other transaction proceeds without the completion of the previous transaction.
- All the transaction operations are interleaved or mixed with each other.
- Non-serial schedules are NOT always recoverable, cascades, strict and consistent.
- In this Schedule, there are two transactions, T1 and T2, executing concurrently. The operations of T1 and T2 are interleaved.
- So, this Schedule is an example of a Non-Serial Schedule.
  Total number of non-serial schedules = Total number of schedules – Total numbe of serial schedules

Image source : Korth

Scheduling

Non-serial Schedule

- In a non-serial Schedule, multiple transactions execute concurrently/simultaneously, unlike the serial Schedule, where one transaction must wait for another to complete all its operations.
- In the Non-Serial Schedule, the other transaction proceeds without the completion of the previous transaction.
- All the transaction operations are interleaved or mixed with each other.
- Non-serial schedules are NOT always recoverable, cascades, strict and consistent.
- In this Schedule, there are two transactions, T1 and T2, executing concurrently. The operations of T1 and T2 are interleaved.
- So, this Schedule is an example of a Non-Serial Schedule.
  Total number of non-serial schedules = Total number of schedules – Total numbe of serial schedules

Image source : Korth

Serialization

- **Basic Assumption** – Each transaction preserves database consistency.
- Thus serial execution of a set of transactions preserves database consistency.
- A (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule. Different forms of schedule equivalence give rise to the notions of:
    1. **conflict Serializability**
    2. **view Serializability**

Serializability

- Simplified view of transactions
  - We ignore operations other than read and write instructions
  - We assume that transactions may perform arbitrary computations on data in local buffers in between reads and writes.
  - Our simplified schedules consist of only read and write instructions.

Conflicting Instructions

- Instructions I_i and I_j of transactions T_i and T_j respectively, **conflict** if and only if there exists some item Q accessed by both I_i and I_j, and at least one of these instructions wrote Q.

1. I_i = **read**(Q), I_j = **read**(Q). I_i and I_j don't conflict.
2. I_i = **read**(Q), I_j = **write**(Q). They conflict.
3. I_i = **write**(Q), I_j = **read**(Q). They conflict
4. I_i = **write**(Q), I_j = **write**(Q). They conflict

Conflict Serializability

- If a schedule *S* can be transformed into a schedule *S´* by a series of swaps of non-conflicting instructions, we say that *S* and *S´* are **conflict equivalent**.
- We say that a schedule *S* is **conflict serializable** if it is conflict equivalent to a serial schedule

Conflict Serializability

- Schedule 3 can be transformed into Schedule 6, a serial schedule where T_2 follows T_1, by series of swaps of non-conflicting instructions.
  - Therefore Schedule 3 is conflict serializable.

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

Schedule 3

T_1	T_2
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

Schedule 6

Image source : Korth

Conflict Serializability

- Example of a schedule that is not conflict serializable:

T_3	T_4
read(Q)	
	write(Q)
write(Q)	

- We are unable to swap instructions in the above schedule to obtain either the serial schedule < T_3, T_4 >, or the serial schedule < T_4, T_3 >.

Other Notions of Serializability

- The schedule below produces same outcome as the serial schedule < T_1, T_5 >, yet is not conflict equivalent or view equivalent to it.
- Determining such equivalence requires analysis of operations other than read and write.

T_1	T_5
read(A)	
$A := A - 50$	
write(A)	
	read(B)
	$B := B - 10$
	write(B)
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$A := A + 10$
	write(A)

Image source : Korth

Testing for Serializability

- Consider some schedule of a set of transactions T_1, T_2, ..., T_n
- **Precedence graph** — a direct graph where the vertices are the transactions (names).
- We draw an arc from T_i to T_j if the two transaction conflict, and T_i accessed the data item on which the conflict arose earlier.
- We may label the arc by the item that was accessed.
- **Example 1**

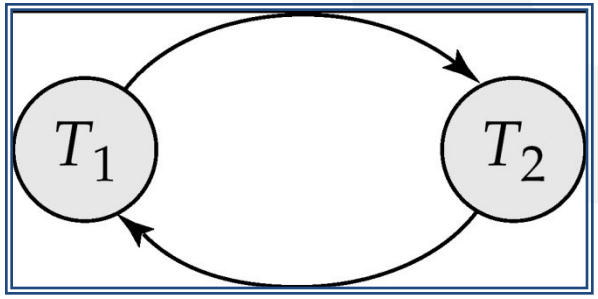


Image source : Google

Example Schedule (Schedule A) + Precedence Graph

T_1	T_2	T_3	T_4	T_5
	read(X)			
read(Y)				
read(Z)				
			read(V)	
			read(W)	
			read(W)	
	read(Y)			
	write(Y)			
		write(Z)		
read(U)				
			read(Y)	
			write(Y)	
			read(Z)	
			write(Z)	
read(U)				

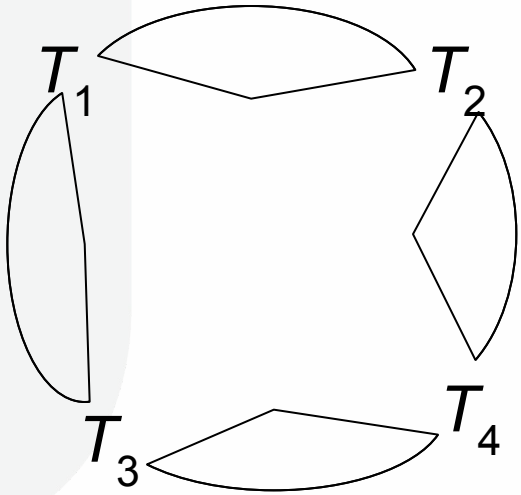


Image source : Google

Test for Conflict Serializability

- A schedule is conflict serializable if and only if its precedence graph is acyclic.
- Cycle-detection algorithms exist which take order n^2 time, where n is the number of vertices in the graph.
  - (Better algorithms take order $n + e$ where e is the number of edges.)
- If precedence graph is acyclic, the Serializability order can be obtained by a *topological sorting* of the graph.
  - This is a linear order consistent with the partial order of the graph.
  - For example, a Serializability order for Schedule A would be

    $T_5 \rightarrow T_1 \rightarrow T_3 \rightarrow T_2 \rightarrow T_4$
    - Are there others?

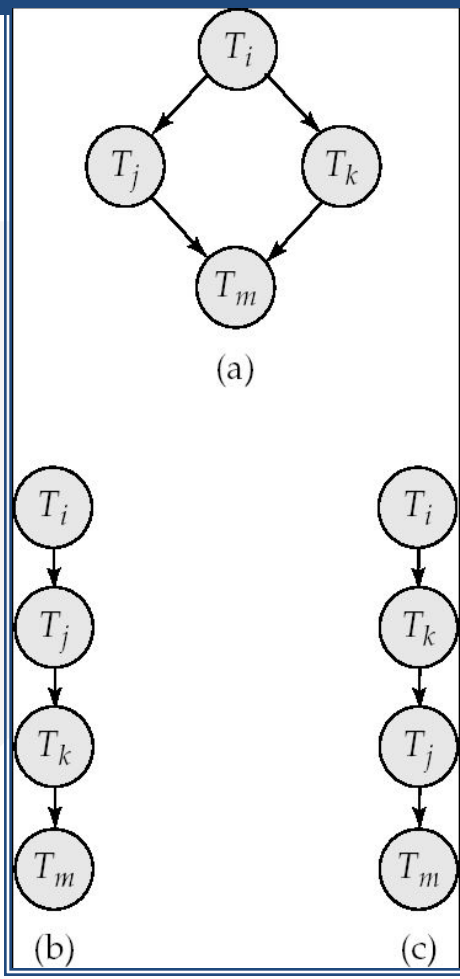


Image source : Google

View Serializability

- Let *S* and *S´* be two schedules with the same set of transactions. *S* and *S´* are **view equivalent** if the following three conditions are met, for each data item *Q,*
  1. If in schedule S, transaction T_i reads the initial value of *Q*, then in schedule *S'* also transaction T_i must read the initial value of *Q.*
  2. If in schedule S transaction T_i executes **read**(*Q*), and that value was produced by transaction T_j (if any), then in schedule *S'* also transaction T_i must read the value of *Q* that was produced by the same **write**(Q) operation of transaction T_j .
  3. The transaction (if any) that performs the final **write**(*Q*) operation in schedule *S* must also perform the final **write**(*Q*) operation in schedule *S'.*
- As can be seen, view equivalence is also based purely on **reads** and **writes** alone.

View Serializability

T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

- Without read , if read perform is ‘Blind Write’
- If there is no blind writes and if schedules is not CS than schedule is not serializable
- If there is blind writes and if schedules is not CS than check for view serializable (VS).
- Every CS is VS but vise versa is not true.
- In view we have to check the sequence of only read.

Image source : Korth

View Serializability

- A schedule *S* is **view serializable** if it is view equivalent to a serial schedule.
- Every conflict serializable schedule is also view serializable.
- Below is a schedule which is view-serializable but *not* conflict serializable.

T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

- Every view serializable schedule that is not conflict serializable has **blind writes.**

Image source : Korth

Test for View Serializability

- The precedence graph test for conflict Serializability cannot be used directly to test for view Serializability.
  - Extension to test for view Serializability has cost exponential in the size of the precedence graph.
- The problem of checking if a schedule is view serializable falls in the class of *NP*-complete problems.
  - Thus existence of an efficient algorithm is *extremely* unlikely.

Parul®
University

DIGITAL LEARNING CONTENT

Database Recovery

Parul®
University

DIGITAL LEARNING CONTENT

Database Recovery

Parul® University

Database Recovery

- DBMS is a extremely complex system where thousands of transactions get executed each second. The resilience and robustness of a DBMS depends on its composite architecture and its underlying hardware and system software. If it fails or crashes during transactions, it is expected that the system would follow some sort of algorithm or techniques to recover lost data.
- There are two types of errors can which leads to transaction failure:-
  - (a) Logical Error:- Code Error or any Query structure error
  - (b) System Error:- Error occurred due to system failure e.g. Deadlock
- System Crash
- Disk Failure

Parul® University

Database Recovery (Continued….)

- **Recovery and Atomicity**
- When a system crashes, there ma be a possibility that some of the transactions are getting executed and various files are opened for them to update the data items. Transactions are made of different operations, which must be atomic in nature. But according to ACID property of DBMS, atomicity must be maintained throughout the transaction.
- Below points must be maintained in case of recovery from failure:-
- It should verify the current state of all the transactions, which were in execution
- A transaction may be in intermediate of some operation; the DBMS must assure the atomicity of the transaction in this case.
- It should verify whether the transaction can be completed now or it needs to be rolled back.

Parul® University

Database Recovery (Continued….)

- **Recovery and Atomicity**
- There are two types of techniques, which can help a DBMS in recovering as well as maintaining the atomicity of a transaction −
- By Maintaining the logs of each transaction, and writing them onto in some storage and then modify the actual data.
- Maintaining shadow paging.

Parul® University

Database Recovery (Continued….)

- **Log-based Recovery**
- Log is a list of records which are logically related, which stores the records of activities carried out by a transaction. It is important that the logs are written before actual modification and stored on a stable storage media, which must be safe.
- 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>

Parul® University

Database Recovery (Continued….)

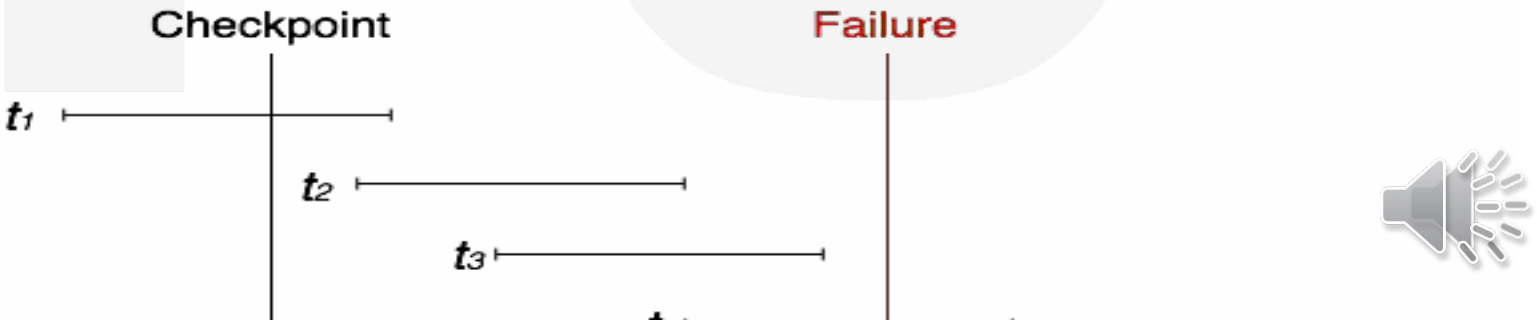
- **Recovery with Concurrent Transactions**
- When some transaction are getting executed in parallel, the logs are interleaved. At the moment of recovery, it will become tough to recover system to restore all logs, and then start recover. To ease this situation, most modern DBMS use the concept of **'checkpoints'.**
- **Checkpoint**
- Checkpoint is a technique 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.

Parul® University

DIGITAL LEARNING CONTENT

Database Recovery (Continued….)

- **Recovery with Concurrent Transactions**
- Transaction behave in following manner when it crashes and then recover:-
- The 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.



Database Recovery

- Recovery is the process of restoring a database to the correct state in the event of a failure.
- It ensures that the database is reliable and remains in consistent state in case of a failure.

Database recovery can be classified into two parts;

1. Rolling Forward applies redo records to the corresponding data blocks.

2. Rolling Back applies rollback segments to the datafiles. It is stored in transaction tables.

We can recover the database using Log–Based Recovery.

Database Recovery

Log-Based Recovery

- Logs are the sequence of records, that maintain the records of actions performed by a transaction.
- In Log – Based Recovery, log of each transaction is maintained in some stable storage. If any failure occurs, it can be recovered from there to recover the database.
- The log contains the information about the transaction being executed, values that have been modified and transaction state.
- All these information will be stored in the order of execution.

Example:
Assume, a transaction to modify the address of an employee. The following logs are written for this transaction,
Log 1: Transaction is initiated, writes 'START' log.
Log: <Tn START>

Database Recovery

Log 2: Transaction modifies the address from 'Pune' to 'Mumbai'.
Log: <T_n Address, 'Pune', 'Mumbai'>

Log 3: Transaction is completed. The log indicates the end of the transaction.
Log: <T_n COMMIT>

There are two methods of creating the log files and updating the database,
1. Deferred Database Modification
2. Immediate Database Modification

1. In Deferred Database Modification, all the logs for the transaction are created and stored into stable storage system. In the above example, three log records are created and stored it in some storage system, the database will be updated with those steps.

Database Recovery

2. In Immediate Database Modification, after creating each log record, the database is modified for each step of log entry immediately.

In the above example, the database is modified at each step of log entry that means after first log entry, transaction will hit the database to fetch the record, then the second log will be entered followed by updating the employee's address, then the third log followed by committing the database changes.

Recovery with Concurrent Transaction

- When two transactions are executed in parallel, the logs are interleaved. It would become difficult for the recovery system to return all logs to a previous point and then start recovering.

To overcome this situation 'Checkpoint' is used

Database Recovery

Checkpoint

- Checkpoint acts like a benchmark.
- Checkpoints are also called as Syncpoints or Savepoints.
- It is a mechanism where all the previous logs are removed from the system and stored permanently in a storage system.
- It declares a point before which the database management system was in consistent state and all the transactions were committed.
- It is a point of synchronization between the database and the transaction log file.
- It involves operations like writing log records in main memory to secondary storage, writing the modified blocks in the database buffers to secondary storage and writing a checkpoint record to the log file.
- The checkpoint record contains the identifiers of all transactions that are

Parul® University

DIGITAL LEARNING CONTENT

Database Recovery

Recovery

- When concurrent transactions crash and recover, the checkpoint is added to the transaction and recovery system recovers the database from failure in following manner,
  1. Recovery system reads the log files from end to start checkpoint. It can reverse the transaction.
  2. It maintains undo log and redo log.
  3. It puts the transaction in the redo log if the recovery system sees a log <Tn, Commit>.
  4. It puts the transaction in undo log if the recovery system sees a log with <Tn, Start>.
- All the transactions in the undo log are undone and their logs are removed.
- All the transactions in the redo log and their previous lc gs are removed and then redone before saving their logs.

Shadow Paging

Shadow paging is one of the techniques that is used to recover from failure. We all know that recovery means to get back the information, which is lost. It helps to maintain database consistency in case of failure.

Concept of shadow paging

Now let see the concept of shadow paging step by step −

- **Step 1** − Page is a segment of memory. Page table is an index of pages. Each table entry points to a page on the disk.
- **Step 2** − Two page tables are used during the life of a transaction: the current page table and the shadow page table. Shadow page table is a copy of the current page table.
- **Step 3** − When a transaction starts, both the tables look identical, the current table is updated for each write operation.

Shadow Paging

- Step 4 − The shadow page is never changed during the life of the transaction.
- Step 5 − When the current transaction is committed, the shadow page entry becomes a copy of the current page table entry and the disk block with the old data is released.
- Step 6 − The shadow page table is stored in non-volatile memory. If the system crash occurs, then the shadow page table is copied to the current page table.

Shadow Paging

Advantages

The advantages of shadow paging are as follows −

- No need for log records.
- No undo/ Redo algorithm.
- Recovery is faster.

Disadvantages

The disadvantages of shadow paging are as follows −

- Data is fragmented or scattered.
- Garbage collection problem. Database pages containing old versions of modified data need to be garbage collected after every transaction.
- Concurrent transactions are difficult to execute.

Recoverable Schedules

- Need to address the effect of transaction failures on concurrently running transactions.
- **Recoverable schedule** — if a transaction T_j reads a data item previously written by a transaction T_i, then the commit operation of T_i appears before the commit operation of T_j.
- The following schedule (Schedule 11) is not recoverable if T_9 commits immediately after the read.
- If T_8 should abort, T_9 would have read (and possibly shown to the user) an inconsistent database state. Hence, database must ensure that schedules are recoverable.

T_8	T_9
read(A)	
write(A)	
	read(A)
read(B)	

Image source : Korth

Cascading Rollbacks

- **Cascading rollback** – a single transaction failure leads to a series of transaction rollbacks. Consider the following schedule where none of the transactions has yet committed (so the schedule is recoverable). If T_{10} fails, T_{11} and T_{12} must also be rolled back.
- Can lead to the undoing of a significant amount of work.

T_{10}	T_{11}	T_{12}
read(A)		
read(B)		
write(A)		
	read(A)	
	write(A)	
		read(A)

Image source : Korth

Cascadeless Schedules

- **Cascadeless schedules** — cascading rollbacks cannot occur; for each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before the read operation of T_j.
- Every cascadeless schedule is also recoverable
- It is desirable to restrict the schedules to those that are cascadeless

Parul®
University

DIGITAL LEARNING CONTENT

Concurrency Control

Concurrency Control

- To ensure serializability practically use concurrency control protocol.
- A database must provide a mechanism that will ensure that all possible schedules are
  - either conflict or view serializable, and
  - are recoverable and preferably cascadeless
- A policy in which only one transaction can execute at a time generates serial schedules, but provides a poor degree of concurrency
  - Are serial schedules recoverable/cascadeless?
- Testing a schedule for serializability *after* it has executed is a little too late!
- **Goal** – to develop concurrency control protocols that will assure serializability.

Locking and Time Stamp Based Schedulers

- Whenever Multiple transactions are executed concurrently , It is exceptionally important to maintain concurrency between all of the concurrent transactions.
- Concurrency Protocols can be used to maintain ACID properties of all transactions and they can be divided into two categories:-

  (i) Lock Based Protocol

  (ii) Time Stamp based Protocol

Parul® University

Lock Based Protocols

- Lock based Protocol provides a mechanism which
- Locks are of two types:-

  (i) Lock Based Protocol

  (ii) Time Stamp based Protocol

Parul® University

Lock Based Protocols (Continued….)

- Lock based Protocols provide a mechanism in which any transaction cannot read or write data without suitable lock.
- Locks are of two types:-

  (i) Binary Locks:- A lock on data items can be in two states only, it will be either locked or unlocked.

  (ii) Shared/exclusive:- This type of locking mechanism differentiates the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is an exclusive lock.

Parul® University

DIGITAL LEARNING CONTENT

Lock Based Protocols (Continued….)

- **There are four types of lock protocols available –**

  (i) Simplistic Lock Protocol

  (ii) Pre-claiming Lock Protocol

  (iii) Two-Phase Locking 2PL

  (iv) Strict Two-Phase Locking

Parul®
University

Lock Based Protocols (Continued….)

(i) Simplistic Lock Protocol :-

It means lock must be obtained before every “write” operation and after completion of “write” operation transaction must release lock

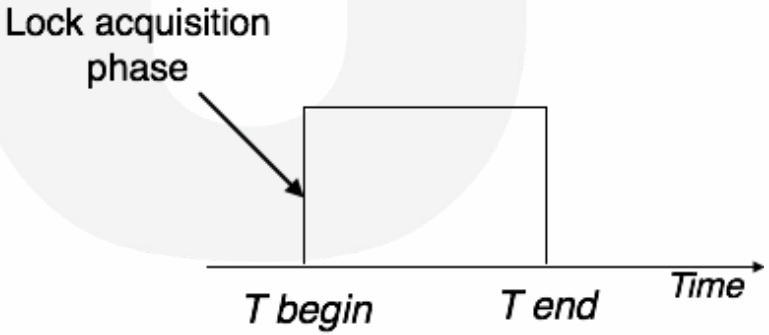
Parul® University

DIGITAL LEARNING CONTENT

Lock Based Protocols (Continued….)

(ii) Pre-claiming Lock Protocol :-

- This Protocol will create a list of database operation and after evaluating them ,Locks will be granted.
- If all locks are granted , After completing all the database operations Transaction will release all the locks.



Parul® University

Lock Based Protocols (Continued….)

(iii) Two-Phase Locking 2PL:-

- According to this protocol Execution phase will get divided into three parts as follows:

(a) Transaction will need permission to get a lock

(b) Transaction will acquire all the locks

After completion when transaction will release its 1st lock then 3rd phase will start

(c) Now Transaction cannot get a new lock it will release the acquired lock.

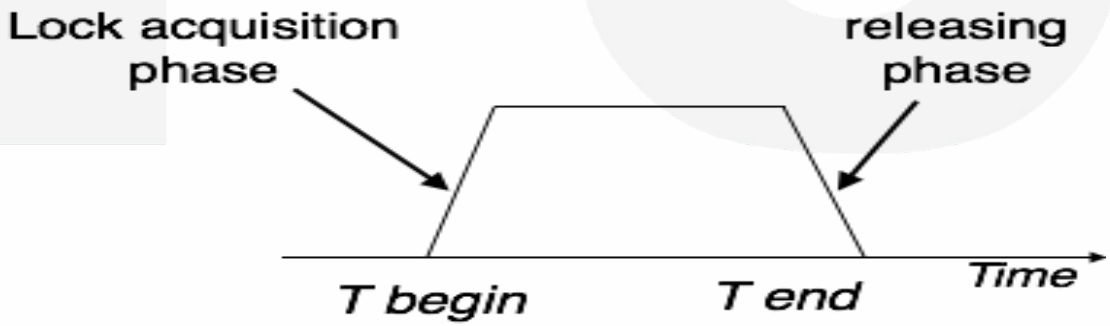
Parul® University

DIGITAL LEARNING CONTENT

Lock Based Protocols (Continued….)

(iii) Two-Phase Locking 2PL (Continued…..)

- This Process of acquiring and releasing lock is called as Growing and Shrinking Phase respectively.
- In **Growing Phase,** Locks will get acquired.
- In **Shrinking Phase,** Locks will get released.



Parul® University

DIGITAL LEARNING CONTENT

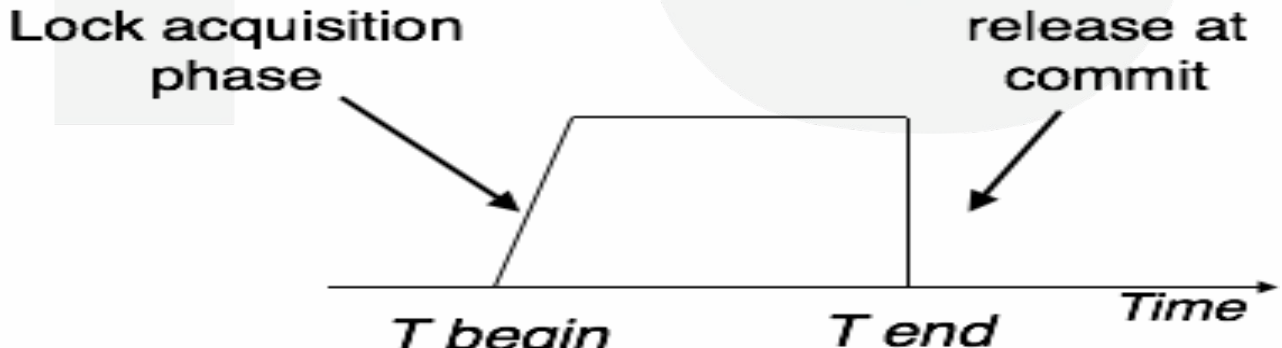
Lock Based Protocols (Continued….)

(iv) Strict Two-Phase Locking :-

•According to this protocol Execution phase will get divided into three parts as follows:

(a) Transaction will need permission to get a lock and will acquire all the locks

(b) Now Transaction is not allowed to release any lock until commit (successful transaction) transaction.



Parul® University

Time Stamp Based Protocols (Continued….)

- The timestamp-ordering protocol assures serializability between transactions in their conflict read and write operations. This is the liability of the protocol system that the conflict pair of tasks should be completed according to the timestamp values of the transactions.
- The timestamp of transaction T_i is denoted as $TS(T_i)$.
- Read time-stamp of data-item X is denoted by R-timestamp(X).
- Write time-stamp of data-item X is denoted by W-timestamp(X).

Parul® University

Time Stamp Based Protocols (Continued….)

- Timestamp ordering protocol works as follows −
- **If a transaction Ti issues a read(X) operation −**
  - If TS(Ti) < W-timestamp(X)
    - Operation rejected.
  - If TS(Ti) >= W-timestamp(X)
    - Operation executed.
  - All data-item timestamps updated.
- **If a transaction Ti issues a write(X) operation −**
  - If TS(Ti) < R-timestamp(X)
    - Operation rejected.
  - If TS(Ti) < W-timestamp(X)
    - Operation rejected and Ti rolled back.
  - Otherwise, operation executed.

Parul®
University

DIGITAL LEARNING CONTENT

Deadlock

Parul® University

Deadlock

- A deadlock is a condition where two or more transactions are waiting indefinitely for one another to give up locks.

For example: In the student table, transaction T1 holds a lock on some rows and needs to update some rows in the grade table. Simultaneously, transaction T2 holds locks on some rows in the grade table and needs to update the rows in the Student table held by Transaction T1.

Now, the main problem arises. Now Transaction T1 is waiting for T2 to release its lock and similarly, transaction T2 is waiting for T1 to release its lock. All activities come to a halt state and remain at a standstill. It will remain in a standstill until the DBMS detects the deadlock and aborts one of the transactions.

Deadlock

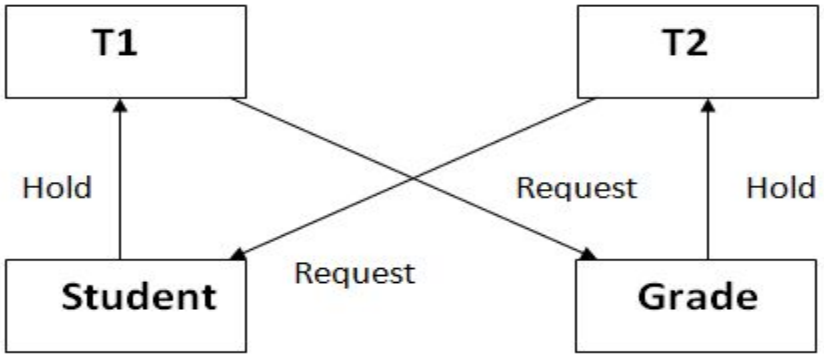


Figure: Deadlock in DBMS

Deadlock Avoidance:

When a database is stuck in a deadlock, It is always better to avoid the deadlock rather than restarting or aborting the database.

The deadlock avoidance method is suitable for smaller databases whereas the deadlock prevention method is suitable for larger databases.

Parul® University

Deadlock

- Example, Transactions that access Students and Grades should always access the tables in the same order. In this way, in the scenario described above,TransactionT1simplywaitsfortransactionT2toreleasethelockon Grades before it begins.
- When transaction T2 releases the lock, Transaction T1 can proceed freely. Another method for avoiding deadlock is to apply both the row-level locking mechanism and the READ COMMITTED isolation level.

Deadlock

Deadlock Detection: When a transaction waits indefinitely to obtain a lock, The database management system should detect whether the transaction is involved in a deadlock or not.

Wait-for-graph is one of the methods for detecting the deadlock situation. This method is suitable for smaller databases. In this method, a graph is drawn based on the transaction and its lock on the resource. If the graph created has a closed loop or a cycle, then there is a deadlock.

For the above-mentioned scenario, the Wait-For graph is drawn below:

Parul® University

Deadlock

Deadlock Prevention

- Deadlock prevention method is suitable for a large database. If the resources are allocated in such a way that deadlock never occurs, then the deadlock can be prevented.
- The Database management system analyzes the operations of the transaction whether they can create a deadlock situation or not. If they do, then the DBMS never allowed that transaction to be executed.

1) **Wait-Die scheme**

In this scheme, if a transaction requests for a resource which is already held with a conflicting lock by another transaction then the DBMS simply checks the timestamp of both transactions. It allows the older transaction to wait until the resource is available for execution.

Parul® University

Deadlock

Let's assume there are two transactions Ti and Tj and let TS(T) is a timestamp of any transaction T. If T2 holds a lock by some other transaction and T1 is requesting for resources held by T2 then the following actions are performed by DBMS:

1. Check if TS(Ti) < TS(Tj) - If Ti is the older transaction and Tj has held some resource, then Ti is allowed to wait until the data-item is available for execution. That means if the older transaction is waiting for a resource which is locked by the younger transaction, then the older transaction is allowed to wait for resource until it is available.
2. Check if TS(T_i) < TS(Tj) - If Ti is older transaction and has held some resource and if Tj is waiting for it, then Tj is killed and restarted later with the random delay but with the same timestamp.

Deadlock

2) Wound wait scheme

- In wound wait scheme, if the older transaction requests for a resource which is held by the younger transaction, then older transaction forces younger one to kill the transaction and release the resource.
- After the minute delay, the younger transaction is restarted but with the same timestamp.
- If the older transaction has held a resource which is requested by the Younger transaction, then the younger transaction is asked to wait until older releases it.

3) Timeouts Based Approach

- To avoid deadlocks caused by indefinite waiting, a timeout mechanism can be used to limit the amount of time a process can wait for a resource.

Parul® University

Deadlock

In this approach, a transaction that has requested a lock waits for at most a specified amount of time. If the lock has not been granted within that time, the transaction is said to time out, and it rolls itself back and restarts.

Features of deadlock in a DBMS:

Mutual Exclusion: Each resource can be held by only one transaction at a time, and other transactions must wait for it to be released.

Hold and Wait: Transactions can request resources while holding on to resources already allocated to them.

No Preemption: Resources cannot be taken away from a transaction forcibly, and the transaction must release them voluntarily.

Parul® University

Deadlock

Circular Wait: Transactions are waiting for resources in a circular chain, where each transaction is waiting for a resource held by the next transaction in the chain.

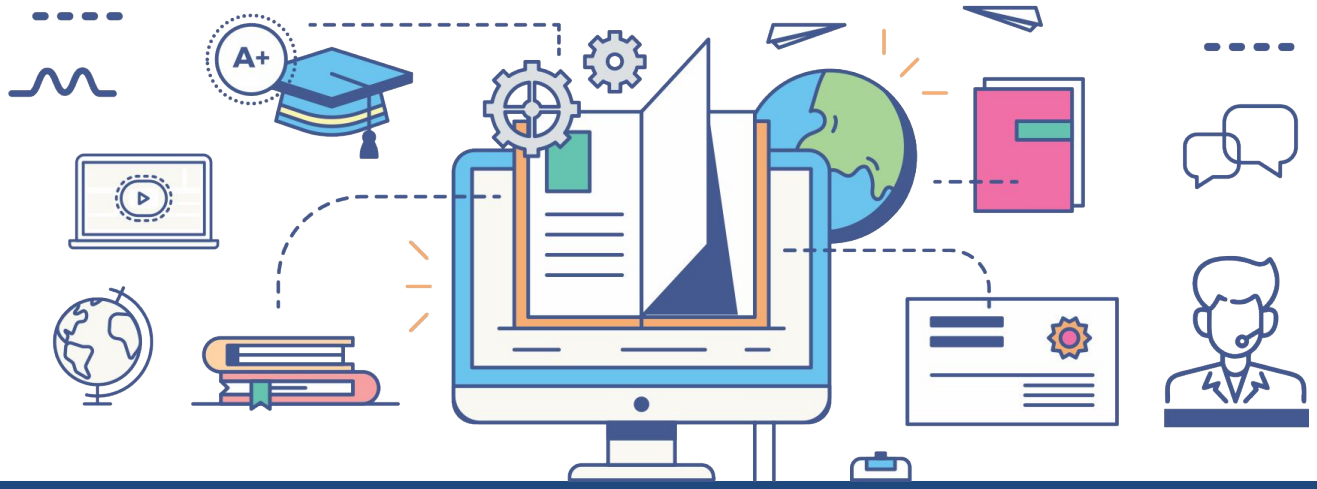
Indefinite Blocking: Transactions are blocked indefinitely, waiting for resources to become available, and no transaction can proceed.

System Stagnation: Deadlock leads to system stagnation, where no transaction can proceed, and the system is unable to make any progress.

Inconsistent Data: Deadlock can lead to inconsistent data if transactions are unable to complete and leave the database in an intermediate state.

Difficult to Detect and Resolve: Deadlock can be difficult to detect and resolve, as it may involve multiple transactions, resources, and dependencies.

DIGITAL LEARNING CONTENT



Parul® University

www.paruluniversity.ac.in