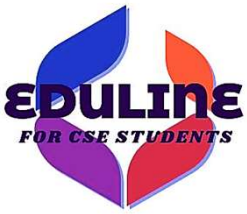
MODULE 5

TRANSACTIONS, CONCURRENCY AND RECOVERY

CO – Students will be able to summarize the transaction concept



Prepared By Mr. EBIN PM

TRANSACTION CONCEPT

- A transaction is a unit of program execution that accesses and possibly updates various data items.
- A transaction is made to change data in a database which can be done by inserting new data, updating the existing data, or by deleting the data that is no longer required.
- There are certain types of transaction states which tell the user about the current condition of that database transaction and what further steps to be followed for the processing.
- Read(X): This read operation is applied to read the X's value from the database server and keeps it in a buffer in the main memory.
- Write(X): This write operation is applied to write the X's value back to the database server from the buffer.

Prepared By Mr.EBIN PM EDULINE 2

Example: transaction to transfer $50 from account A to account B:

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

➢Two main issues to deal with:

- Failures of various kinds, such as hardware failures and system crashes - Recovery
- Concurrent execution of multiple transactions

Transaction to transfer $50 from account A to account B:

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

❖**Atomicity requirement**

- If the transaction fails after step 3 and before step 6, money will be "lost" leading to an inconsistent database state
- Failure could be due to software or hardware
- The system should ensure that updates of a partially executed transaction are not reflected in the database

❖Durability requirement

- once the user has been notified that the transaction has completed (i.e., the transfer of the $50 has taken place), the updates to the database by the transaction must persist even if there are software or hardware failures.

❖Consistency requirement

➢The sum of A and B is unchanged by the execution of the transaction. In general, consistency requirements include

- Explicitly specified integrity constraints such as primary keys and foreign keys
- Implicit integrity constraints–e.g. sum of balances of all accounts, minus sum of loan amounts must equal value of cash-in-hand

Prepared By Mr.EBIN PM EDULINE 5

- A transaction must see a consistent database.
- During transaction execution the database may be temporarily inconsistent.
- When the transaction completes successfully the database must be consistent
- Erroneous transaction logic can lead to inconsistency

❖Isolation requirement

- If between steps 3 and 6, another transaction T2 is allowed to access the partially updated database, it will see an inconsistent database (the sum A + B will be less than it should be).

Prepared By Mr.EBIN PM EDULINE 6

T1	T2
1. **read**(*A*)	
2. *A* := *A* – 50	
3. **write**(*A*)	
	read(A), read(B), print(A+B)
4. **read**(*B*)	
5. *B* := *B* + 50	
6. **write**(*B*	

- Isolation can be ensured trivially by running transactions serially. That is, one after the other.
- However, executing multiple transactions concurrently has significant benefits.

Prepared By Mr.EBIN PM EDULINE 7

ACID PROPERTIES

- A transaction is a unit of program execution that accesses and possibly updates various data items. To preserve the integrity of data the database system must ensure:

1. **Atomicity** - Either all operations of the transaction are properly reflected in the database or none are.
2. **Consistency** - Execution of a transaction in isolation preserves the consistency of the database.
3. **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.

Prepared By Mr.EBIN PM, AP EDULINE 8

- That is, 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.

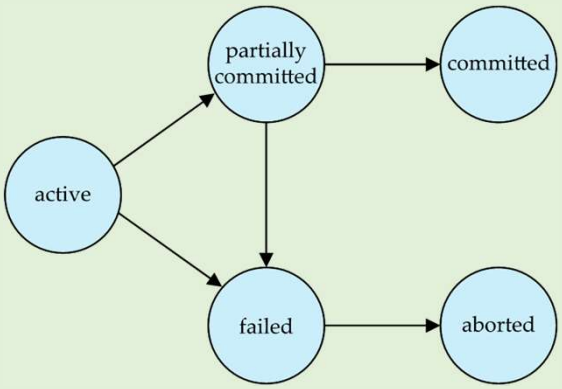
4. **Durability** - After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

TRANSACTION STATE

1. **Active** – the initial state; the transaction stays in this state while it is executing
2. **Partially committed** – after the final statement has been executed.
3. **Failed** -- after the discovery that normal execution can no longer proceed.
4. **Aborted** – after the transaction has been rolled back and the database restored to its state prior to the start of the transaction. Two options after it has been aborted:

- Restart the transaction. It can be done only if no internal logical error
- Kill the transaction

5. **Committed** – after successful completion.



Prepared By Mr.EBIN PM, AP EDULINE 11

IMPLEMENTATION OF ATOMICITY AND DURABILITY USING SHADOW COPY

- In the shadow-copy scheme, a transaction that wants to update the database first creates a complete copy of the database.
- All updates are done on the new database copy, leaving the original copy, the shadow copy, untouched.
- If at any point the transaction has to be aborted, the system merely deletes the new copy.
- The old copy of the database has not been affected.

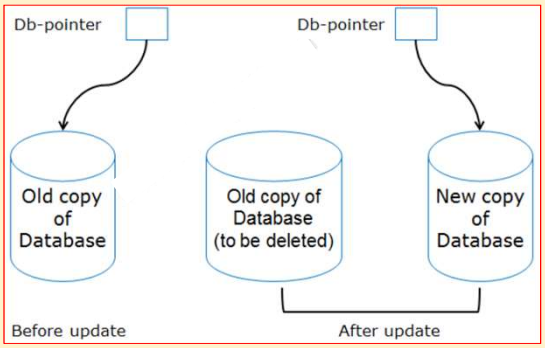
Prepared By Mr.EBIN PM, AP EDULINE 12

- This scheme is based on making copies of the database, called shadow copies, assumes that only one transaction is active at a time.
- The scheme also assumes that the database is simply a file on disk.
- A pointer called db-pointer is maintained on disk; it points to the current copy of the database.

➢If the transaction completes, it is committed as follows:

- First, the operating system is asked to make sure that all pages of the new copy of the database have been written out to disk. (Unix systems use the flush command for this purpose.)

- After the operating system has written all the pages to disk, the database system updates the pointer db-pointer to point to the new copy of the database; the new copy then becomes the current copy of the database. The old copy of the database is then deleted.
- The transaction is said to have been committed at the point where the updated db pointer is written to disk.



CONCURRENT EXECUTION

- Transaction-processing systems usually allow multiple transactions to run concurrently.
- Allowing multiple transactions to update data concurrently causes several complications with consistency of the data.
- Ensuring consistency in spite of concurrent execution of transactions requires extra work; it is far easier to insist that transactions run serially—that is, one at a time, each starting only after the previous one has completed.
- However, there are two good reasons for allowing concurrency:

➢Improved throughput and resource utilization:

- A transaction consists of many steps. Some involve I/O activity; others involve CPU activity. The CPU and the disks in a computer system can operate in parallel. Therefore, I/O activity can be done in parallel with processing at the CPU.
- The parallelism of the CPU and the I/O system can therefore be exploited to run multiple transactions in parallel.
- While a read or write on behalf of one transaction is in progress on one disk, another transaction can be running in the CPU, while another disk may be executing a read or write on behalf of a third transaction.
- All of this increases the throughput of the system—that is, the number of transactions executed in a given amount of time.

➢**Reduced waiting time:**

- There may be a mix of transactions running on a system, some short and some long.
- If transactions run serially, a short transaction may have to wait for a preceding long transaction to complete, which can lead to unpredictable delays in running a transaction.
- If the transactions are operating on different parts of the database, it is better to let them run concurrently, sharing the CPU cycles and disk accesses among them.
- Concurrent execution reduces the unpredictable delays in running transactions.
- Moreover, it also reduces the average response time: the average time for a transaction to be completed after it has been submitted.

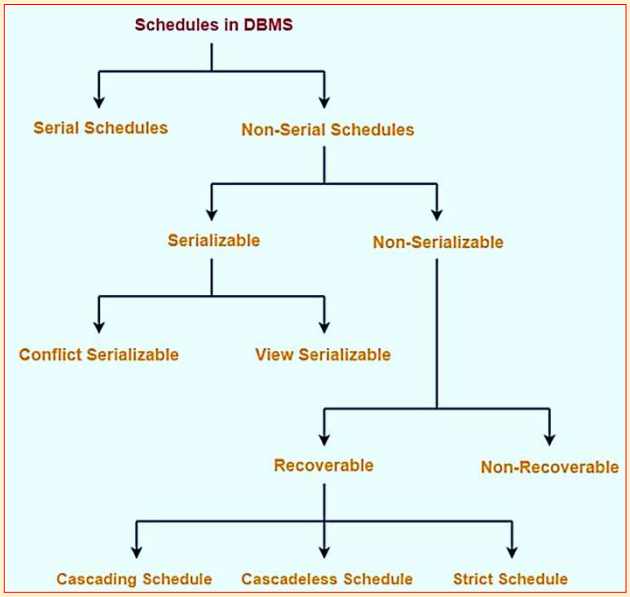
- The database system must control the interaction among the concurrent transactions to prevent them from destroying the consistency of the database. It is achieved using concurrency-control schemes.

SCHEDULE

- A series of operation from one transaction to another transaction is known as schedule.
- It is used to preserve the order of the operation in each of the individual transaction.
- A schedule is the arrangement of transaction operations.
- A schedule may contain a set of transactions.
- A transaction is a set of operations. To run transactions concurrently, we arrange or schedule their operations in an interleaved fashion.

Prepared By Mr.EBIN PM, AP EDULINE 19

- In DBMS, schedules may be classified as



Prepared By Mr.EBIN PM, AP EDULINE 20

➢Serial Schedules

- All the transactions execute serially one after the other.
- When one transaction executes, no other transaction is allowed to execute.
- Serial schedules are **always** consistent, recoverable, cascadeless and strict.
- 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.
- A serial schedule always gives the correct result.

Examples:

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

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

- For a set of n transactions, there exist **n!** different valid serial schedules.
- If there exist three transactions then all possible serial schedules are:

  here n = 3(no of transactions)

  3! = 3*2*1 = 6 serial schedules

2 Transaction	3 Transaction
T1->T2	T1->T2->T3
T2->T1	T1->T3->T2
	T2->T1->T3
	T2->T3->T1
	T3->T1->T2
	T3->T2->T1

➢Non-Serial Schedules

- Multiple transactions execute concurrently.
- Operations of all the transactions are inter leaved or mixed with each other.
- Non-serial schedules are **not always** consistent, recoverable, cascadeless and strict
- 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 further categorized into serializable and non-serializable schedules

Example:

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

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

Prepared By Mr.EBIN PM, AP EDULINE 25

➢ Total Number of Serial Schedules

Total number of serial schedules

= Number of different ways of arranging n transactions

= **n!**

➢ Total Number of Non-Serial Schedules

Total number of non-serial schedules

= **Total number of schedules – Total number of serial schedules**

Prepared By Mr.EBIN PM, AP EDULINE 26

SERIALIZABILITY

- Some non-serial schedules may lead to inconsistency of the database.
- Serializability is a concept that helps to identify which non-serial schedules are correct and will maintain the consistency of the database.
- A serializable schedule always leaves the database in a consistent state.
- A serial schedule is always a serializable schedule because, in a serial Schedule, a transaction only starts when the other transaction has finished execution

Difference between Serial Schedule and Serializable Schedule

Serial Schedules	VS	Serializable Schedules
No concurrency is allowed. Thus, all the transactions necessarily execute serially one after the other.		Concurrency is allowed. Thus, multiple transactions can execute concurrently.
Serial schedules lead to less resource utilization and CPU throughput.		Serializable schedules improve both resource utilization and CPU throughput.
Serial Schedules are less efficient as compared to serializable schedules. (due to above reason)		Serializable Schedules are always better than serial schedules. (due to above reason)

❖Conflict Serializability

- If a given non-serial schedule can be converted into a serial schedule by swapping its non-conflicting operations, then it is called as a conflict serializable schedule.

➢Two operations are called as conflicting operations if all the following conditions hold true for them-

- Both the operations belong to different transactions
- Both the operations are on the same data item
- At least one of the two operations is a write operation

➢Let's consider two different transactions T_i and T_j Then considering the above conditions, the following table follows:

Transaction i	Transaction j	IsConflicting
Readi (X)	Readj (X)	Non-conflicting
Readi (X)	Writej (X)	Conflicting
Writei (X)	Readj (X)	Conflicting
Writei (X)	Writej (X)	Conflicting

Example 1:

Transaction 1	Transaction 2
R1(A)	W2(B)
W1(A)	
	R2(A)
R1(B)	W2(A)

- W1(A) and R2(A) are conflicting operations, because
- W1(A) and R2(A) are part of different transactions.
- Both apply to the same data item, i.e., A.
- W1(A) is a write operation.

- Similarly, W1(A) and W2(A) are conflicting operations as they are part of different transactions working on the same data item, and one of them is the write operation.
- W1(A) and W2(B) are non-conflicting operations as they work on different data items and thus do not satisfy all the given conditions.
- R1(A) and R2(A) are non-conflicting operations as none of them is a write operation and thus does not satisfy the third condition.
- W1(A) and R1(A) are non-conflicting as they belong to the same transactions and thus do not satisfy the first condition.

Example1:

Transaction T1	Transaction T2
R1 (A)	
W1 (A)	
	R2 (A)
R1 (B)	

➢In this schedule,

- W1 (A) and R2 (A) are called as conflicting operations.
- This is because all the above conditions hold true for them.

❖**Conflict Equivalent**

- If a schedule gets converted into another schedule by swapping the non-conflicting operations, they are said to be conflict equivalent schedules.

PRECEDENCE GRAPH TO CHECK CONFLICT SERIALIZABLE SCHEDULE

Algorithm:

1. Create the number of node in the graph equal to the number of transactions in the given schedule.
2. Starting with each and every transaction identify all the existing conflicting operations and represent them in the graph in the form of edges following the direction of the conflicting operation.
3. Check if the precedence graph has either a cycle or a loop.
4. If the cycle or loop does exist, then the given schedule is not conflict serializable.

5. Else the schedule is conflict serializable.
6. In case the schedule is conflict serializable then apply the Topological ordering in the graph to find out the equivalent serial schedule.

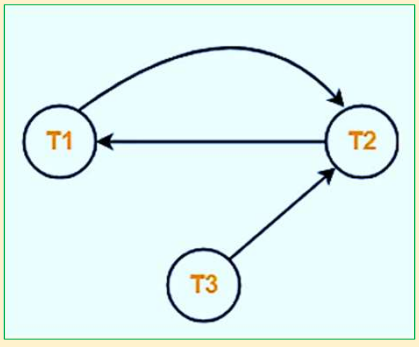
Example 1:

Check whether the given schedule S is conflict serializable or not-

S : R1(A) , R2(A) , R1(B) , R2(B) , R3(B) , W1(A) , W2(B)

1. List all the conflicting operations and determine the dependency between the transactions-

R2(A) , W1(A) (T2 → T1)
R1(B) , W2(B) (T1 → T2)
R3(B) , W2(B) (T3 → T2)



2. Draw the precedence graph

- Clearly, there exists a cycle in the precedence graph.
- Therefore, the given schedule S is not conflict serializable.

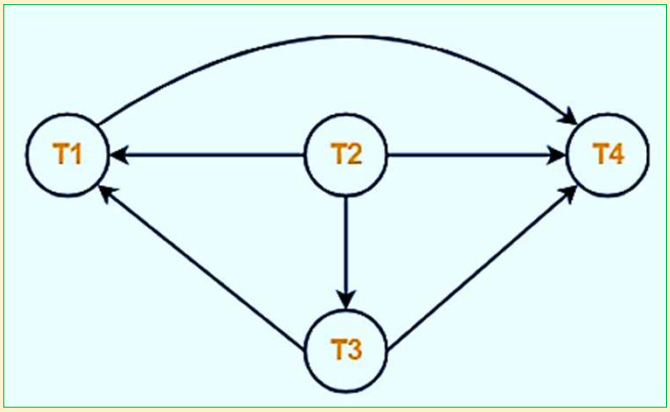
Example2: Check whether the given schedule S is conflict serializable or not?

T1	T2	T3	T4
	R(X)		
		W(X)	
		Commit	
W(X)			
Commit			
	W(Y)		
	R(Z)		
	Commit		
			R(X)
			R(Y)
			Commit

- List all the conflicting operations and determine the dependency between the transactions-

R2(X) , W3(X) (T2 → T3)
R2(X) , W1(X) (T2 → T1)
W3(X) , W1(X) (T3 → T1)
W3(X) , R4(X) (T3 → T4)
W1(X) , R4(X) (T1 → T4)
W2(Y) , R4(Y) (T2 → T4)

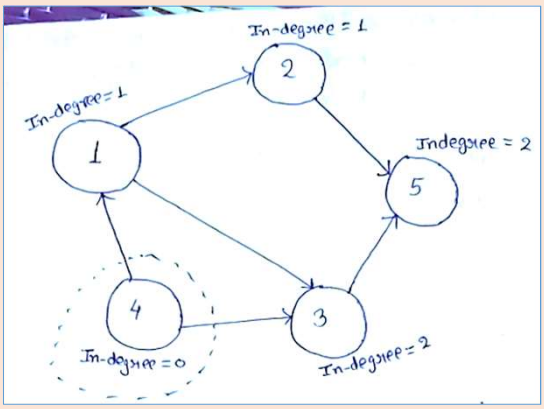
- Draw the precedence graph



- Clearly, there exists no cycle in the precedence graph.
- Therefore, the given schedule S is conflict serializable.

TOPOLOGICAL ORDERING TO FIND EQUIVALENT SERIAL SCHEDULE

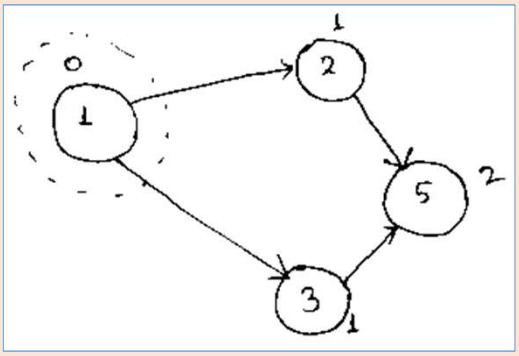
- If the schedule is conflict serializable then apply the Topological ordering in the graph to find out the equivalent serial schedule.
- **Topological Ordering:** The process of selecting the nodes with in-degree zero always, is known as topological ordering.
- while applying this process, after selecting the node with in-degree zero, we shall ignore that node further and also we should ignore the edges that were emerging from that selected node.
- From the remaining graph we should continue finding the next node with in-degree zero until all the nodes of graph gets selected.



- According to topological ordering definition, node 4 has in-degree 0. Hence start from node 4 and select it first.

4				

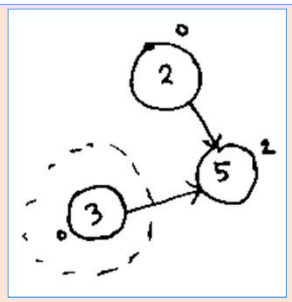
- Now remove node 4 and the edges associated with it.



- Now node 1 has in-degree 0. Hence we will select this node.

4	1			

➢Repeat the steps



- Now node 2 and node 3 both have in-degree 0. Hence we can take any node.

4	1	2		

OR

4	1	3		

Prepared By Mr.EBIN PM, AP, CU EDULINE 43

➢Again repeat

- Hence we will get two result with topological ordering

4	1	2	3	5

OR

4	1	3	2	5

Prepared By Mr.EBIN PM, AP, CU EDULINE 44

VIEW SERIALIZABILITY

- View Serializability is a process used to check whether the given schedule is view serializable or not.
- To do so we check if the given schedule is View Equivalent to its serial schedule.
- There is a possibility that even though the schedule does not conflict serializable, it gives a consistent result
- This is because conflict serializability is limited to the precedence graph of a schedule. If the precedence graph contains any loops/cycles we cannot predict whether a schedule is consistent or inconsistent.

- If its precedence graph doesn't contain any loop/cycle given schedule is consistent, but if it contains a loop/cycle then the schedule may or may not be consistent.
- To figure out the state of the schedule, we use the concept of View-Serializability
- If a given schedule is found to be view equivalent to some serial schedule, then it is called as a view serializable schedule.

View Equivalent

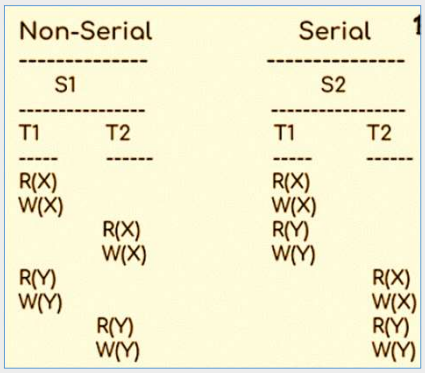
- Lets learn how to check whether the two schedules are view equivalent.
- Two schedules T1 and T2 are said to be view equivalent, if they satisfy all the following conditions:

1. **Initial Read:** Initial read of each data item in transactions must match in both schedules. For example, if transaction T1 reads a data item X before transaction T2 in schedule S1 then in schedule S2, T1 should read X before T2. Here initial read means the first read operation on a data item

2. **Final Write:** Final write operations on each data item must match in both the schedules. For example, a data item X is last written by Transaction T1 in schedule S1 then in S2, the last write operation on X should be performed by the transaction T1.

3. **Update Read:** If in schedule S1, the transaction T1 is reading a data item updated by T2 then in schedule S2, T1 should read the value after the write operation of T2 on same data item. For example, In schedule S1, T1 performs a read operation on X after the write operation on X by T2 then in S2, T1 should read the X after T2 performs write on X.

Example:



- S2 is the serial schedule of S1. Lets check the three conditions of view serializability:

Initial Read

- In schedule S1, transaction T1 first reads the data item X. In S2 also transaction T1 first reads the data item X.
- Lets check for Y. In schedule S1, transaction T1 first reads the data item Y. In S2 also the first read operation on Y is performed by T1.
- The initial read condition is satisfied in S1 & S2.

Final Write

- In schedule S1, the final write operation on X is done by transaction T2. In S2 also transaction T2 performs the final write on X.
- Lets check for Y. In schedule S1, the final write operation on Y is done by transaction T2. In schedule S2, final write on Y is done by T2.
- The final write condition is satisfied in S1 & S2.

Update Read

- In S1, transaction T2 reads the value of X, written by T1. In S2, the same transaction T2 reads the X after it is written by T1.
- In S1, transaction T2 reads the value of Y, written by T1. In S2, the same transaction T2 reads the value of Y after it is updated by T1.
- The update read condition is also satisfied for both the schedules.

➢S1 and S2 are view equivalent

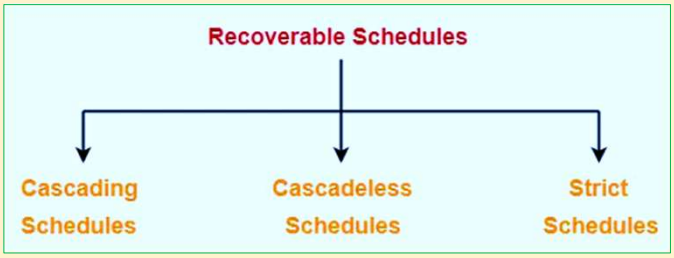
➢The schedule S2 is the serial schedule of S1, thus we can say that the schedule S1 is view serializable schedule.

RECOVERABILITY

- If any transaction that performs a dirty read operation from an uncommitted transaction and also its committed operation becomes delayed till the uncommitted transaction is either committed or rollback such type of schedules is called as Recoverable Schedules.
- If T1 and T2 are two transactions and T2 reads data from T1 then T2 should commit only after T1 commits.
- Mathematically,

$$(T1 \rightarrow T2\ Reads) \Longrightarrow (T1\ Commits \rightarrow T2\ Commits)$$

❖Types of Recoverable Schedules



1. Cascading Schedule

- If in a schedule, failure of one transaction causes several other dependent transactions to rollback or abort, then such a schedule is called as a Cascading Schedule or Cascading Rollback or Cascading Abort.
- It simply leads to the wastage of CPU time.

Prepared By Mr.EBIN PM, AP EDULINE 53

Example:

T1	T2	T3	T4
R (A)			
W (A)			
	R (A)		
	W (A)		
		R (A)	
		W (A)	
			R (A)
			W (A)
Failure			

Prepared By Mr.EBIN PM, AP EDULINE 54

Here,

- Transaction T2 depends on transaction T1.
- Transaction T3 depends on transaction T2.
- Transaction T4 depends on transaction T3.

In this schedule,

- The failure of transaction T1 causes the transaction T2 to rollback.
- The rollback of transaction T2 causes the transaction T3 to rollback.
- The rollback of transaction T3 causes the transaction T4 to rollback.

➢ Such a rollback is called as a Cascading Rollback.

➢ If the transactions T2, T3 and T4 would have committed before the failure of transaction T1, then the schedule would have been irrecoverable.

2. Cascadeless Schedule

- If in a schedule, a transaction is not allowed to read a data item until the last transaction that has written it is committed or aborted, then such a schedule is called as a Cascadeless Schedule.

➢ In other words,

- Cascadeless schedule allows only committed read operations.
- Therefore, it avoids cascading roll back and thus saves CPU time.

Example:

- **Cascade less Schedule**

T1	T2	T3
R (A)		
W (A)		
Commit		
	R (A)	
	W (A)	
	Commit	
		R (A)
		W (A)
		Commit

- Cascadeless schedule allows only committed read operations.
- However, it allows uncommitted write operations.

T1	T2
R (A)	
W (A)	
	W (A) // Uncommitted Write
Commit	

Cascade less Schedule

LOCK BASED PROTOCOLS

- Transaction processing systems usually allow multiple transactions to run concurrently.
- By allowing multiple transactions to run concurrently will improve the performance of the system in terms of increased throughout or improved response time, but this allows causes several complications with consistency of the data.
- Ensuring consistency in spite of concurrent execution of transaction require extra work, which is performed by the concurrency controller system of DBMS.

❖Lock-Based Protocol

- A lock is kind of a mechanism that ensures that the integrity of data is maintained. It does that, by locking the data while a transaction is running, any transaction cannot read or write the data until it acquires the appropriate lock.
- In this type of protocol, any transaction cannot read or write data until it acquires an appropriate lock on it.
- There are two types of lock:

➢**Shared lock (S)**

➢**Exclusive lock (X)**

➢**Shared lock:**

- It is also known as a Read-only lock. (can read but cannot write)
- 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.
- Shared lock is placed when we are reading the data, multiple shared locks can be placed on the data but when a shared lock is placed no exclusive lock can be placed.
- To understand the lock mechanism let's take an example of conflict:

- You and your brother have a joint bank account, from which you both can withdraw money. Now let's say you both go to different branches of the same bank at the same time and try to withdraw 5000 INR, your joint account has only 6000 balance.
- Now if we don't have concurrency control in place you both can get 5000 INR at the same time but once both the transactions finish the account balance would be -4000 which is not possible and leaves the database in inconsistent state.
- We need something that controls the transactions in such a way that allows the transaction to run concurrently but maintaining the consistency of data to avoid such issues.

➢Exclusive lock:

- 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.
- Exclusive lock is placed when we want to read and write the data. This lock allows both the read and write operation, Once this lock is placed on the data no other lock (shared or Exclusive) can be placed on the data until Exclusive lock is released.
- For example, when a transaction wants to update the Steve's account balance, let it do by placing X lock on it but if a second transaction wants to read the data(S lock) don't allow it, if another transaction wants to write the data(X lock) don't allow that either.

➢Lock Compatibility Matrix

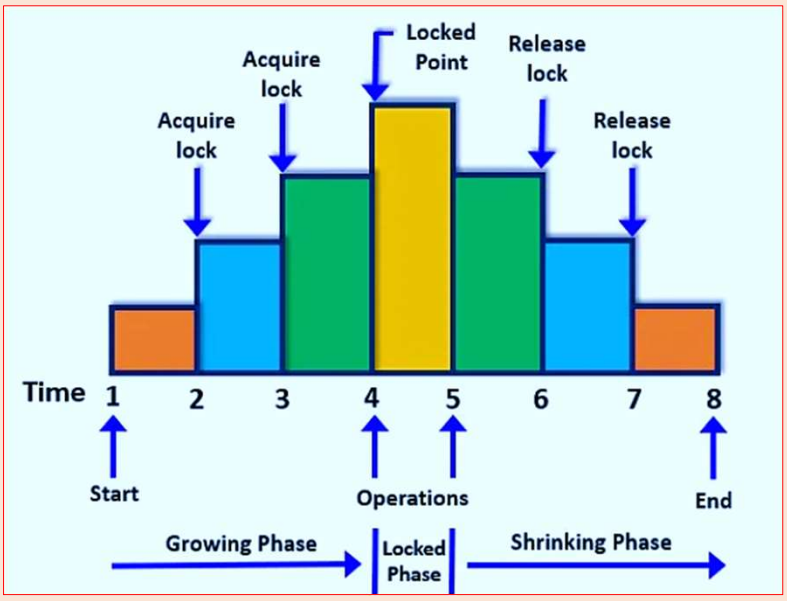
	S	X
S	True	False
X	False	False

- There are two rows, first row says that when S lock is placed, another S lock can be acquired so it is marked true but no Exclusive locks can be acquired so marked False.
- In second row, When X lock is acquired neither S nor X lock can be acquired so both marked false.

TWO PHASE LOCKING PROTOCOL (2PL)

- In two phase locking protocol the locking and unlocking of data items is done in two phases.
- **Growing Phase:** In this phase, the locks are acquired on the data items but none of the acquired locks can be released in this phase.
- **Shrinking Phase:** The existing locks can be released in this phase but no new locks can be acquired in this phase.
- The above phases in a DBMS are determined by something called a '**Lock Point**'. Lock point is the point where a transaction has achieved its final lock. It is also the point where the growing phase ends and the shrinking phase begins.

Prepared By Mr.EBIN PM EDULINE 65



Prepared By Mr.EBIN PM, AP EDULINE 66

- In the following example there are two transaction T1 and T2 running concurrently.

	T1	T2
Step 1	lock-S(A)	
Step 2	..	lock-S(A)
Step 3	lock-S(B)	
Step 4	...	lock-S(B)
Step 5	lock-X(C)	
Step 6	..	
Step 7	Unlock(A)	
Step 8	Unlock(B)	
Step 9	Unlock(C)	
Step 10		lock-S(C)
Step 11		Unblock(A)
Step 12		Unblock(B)
Step 13		Unblock(C)

- Transaction T1: In this example, growing phase of T1 is from Step 1 to Step 5. Shrinking phase is from Step 7 to Step 9. Lock point is at step 5.
- Transaction T2: Growing phase of T2 is from Step 2 to Step 10. Shrinking phase is from Step 11 to Step 13. Lock point is at step 10.

➢ **Strict Two Phase Locking Protocol (Strict – 2PL)**

- It is somewhat similar to 2PL except that it doesn't have a shrinking phase. This protocol releases all the locks only after the transaction is completed successfully and used the commit statement to make the changes permanent in the database.
- It doesn't release locks after performing an operation on data items. It releases all the locks at the same time once the transaction commit successfully.

LOG-BASED RECOVERY

- When a transaction fails, it is important to rollback the transaction so that changes made by failed transaction doesn't store in the database, this is important to maintain the integrity of database.
- Log is a sequence of records that is maintained in a stable storage devices to note down all the changes made by transactions in a sequential manner.
- This log is used to recover the transaction in case of failure.
- Any operation performed by transaction on database is recorded in the log.

- It is important to record the log before the actual operation performed on the database, this make sure that if an operation fail, it is already recorded in the log.

➢ **How the logs are maintained**

- A transaction T1 is modifying the Department of an employee, for this operation, the following log is maintained:
  - Log entry to mark the start of the transaction:

    <T1, Start>
  - Just before the transaction modifies the department of the employee from "Sales" To "Marketing", the following log is maintained:

    <T1, Department, 'Sales', 'Marketing' >

- Log entry to mark the successful end of the transaction:

  <T1, Commit>

- There are two database modification approaches used by the transactions.

1. Deferred Database Modification

- In this approach, the transaction does not commit the changes the database, until it is completed successfully.
- In this approach, all the logs are created at once and stored in the database.

2. Immediate Database Modification

- In this approach, the transaction make change immediately after an operation is performed by the transaction.

- In this approach, logs are recorded just before the transaction is going to perform an operation in database.

➢Recovery using Log Records

- In case of a transaction failure, the log is referenced to recover the transaction and rollback or redone all the changes done by the transaction.
- If the log contains the entry <Tn, Start> and <Tn, Commit> or <Tn, Start> and <Tn, Abort> then the transaction Tn needs to be redone based on the log entries for each operation recorded in the log.
- If the log contains the entry <Tn, Start> but doesn't contain an entry for <Tn, Commit> or <Tn, Abort> then the transaction needs to be rolled back.

❖Log-based recovery uses the following term for execution as follows.

➢**Transaction Identifier:** It used to uniquely identify the transaction.

➢**Data item Identifier:** It is used to uniquely identify the data item written . Typically, it is the location on disk of the data item.

➢ **Old Value:** It is the value of data before the write operation of a transaction.

➢**New Value:** It is the value of data after the write operation of a transaction.

❖For recovery purposes, we use the following two operations as follows.

➢**Undo (TRX):** This command is used to restore all records updated by transactions to the old value.

➢**Redo (TRX):** This command is used to set the value of all records updated by a transaction to the new value.

Deferred modification	Immediate modification
1. This scheme is easier to implement as fewer operations are needed. Only redo operation is required after a system failure.	Both redo and undo operations are required after a system failure.
2. No extra I/O operations are needed before commit time.	There could be extra I/O operations by operating system to flush out the block buffer.
3. It does not require old values of data item on log.	Both old and new values of data items are written on log.
4. If a transaction requires to read a data item modified by other transaction, it cannot do as its changes may not have gone to the database.	Due to immediate modification a transaction will get the modified value of a data item. This will mean higher concurrency.
5. Write locks are held much longer. Theses locks are held till the commit point. This will lead to a lower concurrency.	Write locks can be released after modification and hence a higher concurrency.
6. For long transaction the memory needed for log and local variables could be very high.	It can manage with less memory space.

CHECKPOINT

- Checkpoint is like a bookmark in the transaction that helps us rollback a transaction till a certain point.
- These are really useful when a transaction performs several operations. If such transaction fail at any point of time, instead of undoing the whole transaction, we can rollback to a certain checkpoint.
- We can have more than one checkpoint in a transaction.
- These checkpoints can be given any name so it's easier to identify the particular point in the transaction and rollback to a certain point in case of failure.

- You can say that by using checkpoints, you can divide the transaction in smaller parts.
- Once a checkpoint is reached, the changes are made permanent in the database till that point and the log entries are removed.
- This is because that part of the transaction is successfully completed so there is no need to roll back or redone, thus no need to maintain those logs.
- A checkpoint represents a point till which all transactions are completed and database is in consistent state.

❖**Recovery using Checkpoint**

- The recovery system reads the log file in reverse (from end to start).

- Recovery system maintains two files: one is redo-list file and second is undo-list file. One or both of these files are used to recover a failed transaction.
- If the recovery system finds a log entry with <Tn, Start> and <Tn, Commit> or just <Tn, Commit>, it puts the transaction in the redo-list. This is because a commit statement represents that some of transactions in this schedule are made permanent using commit statement, so it becomes important to redone the failed transactions.
- If the recovery system finds a log entry with <Tn, Start> but no entry with <Tn, commit> or <Tn, Abort> , it puts the transaction in undo-list. This is because no transaction made the changes permanent in the database as no commit statements found, in this case the transaction can be rolled back by putting it in undo-list.

❖Different Types of Checkpoint

1. Automatic Checkpoint

- Every time each database without a user-defined recovery time, the SQL server database engine generates automatically checkpoints.
- The advanced recovery server provides maximum time to recover a database during the system restart.
- Automatic checkpoint depends on the number of log files generated in the database.
- After a system crash, the recovery time depends on the amount of time required to redo a dirty page which is more than recovery server time.

2. Indirect Checkpoints

- In indirect checkpoint, recovery time is maintained at SQL server database engine and it provides more accurate recovery time as compared to the automatic checkpoint that means a number of dirty pages is less as a compared threshold value in the database.
- In indirect checkpoint, dirty pages in the database are written smoothly in the background.
- From SQL server 2016 the default checkpoint type is an indirect checkpoint and the default recovery time is 60 sec for the created database.
- Physically we can easily transfer the data page in indirect checkpoints.

3. Internal Checkpoints

- An internal checkpoint is used many times to take a backup of the database.
- It is also used to add databases, remove database files, and clean SQL servers.

4. Manual Checkpoints

- This is an optional checkpoint provided in DBMS to provide internal time manually by using checkpoint T-SQL command.
- If checkpoint interval is not specified then a manual checkpoint will be run for completion, the required interval time depends on dirty pages that operation writes.