LET'S START WITH DBMS :)

Transaction And Concurrency control

Transaction is a logical unit of work that comprises one or more database operations(like Read/write/commit/rollback) . In a transaction both read and write operations are fundamental actions that ensure ACID properties of transactions (data consistency and integrity)

Read(R)-> A read operation involves retrieving/fetching data from the database.

Write(W)->A write operation in a transaction involves modifying data in the database

$$\begin{tikzcd}
\overset{1}{\text{Swiggy}} \arrow[r] & \overset{2}{\text{Order}} \arrow[r] & \overset{3}{\text{Payment page}} \arrow[r] & \overset{4}{\text{enter bank details}} \arrow[r] & \overset{5}{\text{otp}} \arrow[r] \arrow[d] & \overset{6}{\text{network failure /issues}} \arrow[r] & \overset{7}{\text{rollback}} \\
 & & & \underset{7}{\text{Order done}} & \underset{6}{\text{No network failure}} \arrow[l] & &
\end{tikzcd}$$

LET'S START WITH DBMS :)

Consider an example of a banking application where a customer Ram wants to transfer 100rs to Shyam

Step 1: Begin Transaction
Step 2: The application will fetch/read the current bal of the Ram-> **R(Ram)**
Step 3: The application will calculate the new balances after the transfer -> **Ram=Ram-100**
Step 4: Update it in the temporary Storage/transaction logs-> **W(Ram)**
Step 5: The application will fetch/read the current bal of the Shyam-> **R(Shyam)**
Step 6: The application will calculate the new balances after the transfer-> **Shyam=Shyam+100**
Step 7 : Update it in the temporary Storage /transaction logs-> **W(Shyam)**
Step 8: If all updates are successful and there are no errors, commit the transaction to make the changes permanent. Logs are maintain till the occurence of commit, after that log files are deleted and database is updated till step7 ->**COMMIT**
Step 9: If an error occurs during any step rollback the transaction to revert changes made within the transaction.-> **ROLLBACK**

LET'S START WITH DBMS :)

Transaction states

BEGIN
R(Ram)
Ram=Ram-100
W(Ram)
R(Shyam)
Shyam=Shyam+100
W(Shyam)
COMMIT
ROLLBACK

Case-1

ACTIVE

PARTIALLY COMMITED
SUCCESSFUL

COMPLETED

Case-2

ACTIVE

FAILED
CANCELLED/ROLLBACK
COMPLETED

LET'S START WITH DBMS :)

Transaction And Concurrency control

Concurrency control ensures that multiple transactions can run concurrently without compromising data consistency.

Example : Consider a banking system where two transactions are happening concurrently

1. Ram giving Shyam 100rs
2. Shyam giving Ram 50rs , the data should be consistent for both transactions

Some techniques used here are:

- Locking
- Two-Phase Locking (2PL)
- Timestamp Ordering

LET'S START WITH DBMS :)

ACID Properties

ACID properties are the properties which ensures that transactions are processed reliably and accurately, even in complex situations(sytem failures/network issues)

- A-> Atomicity (Either execute all operations or none)
- C-> Consistency (Read should fetch upto date data and write shouldn't voilate integrity constraints)
- I-> Isolation (One transaction should be independent of other transaction)
- D-> Durability (The commited transaction should remain even after a failure/crash)

LET'S START WITH DBMS :)

ACID Properties

A-> Atomicity: It ensures that a transaction is treated as a single unit of work
Either all operations are completed successfully (commit) or none of them are applied (rollback).

This guarantees that the database remains in a consistent state despite any failures or interruptions during the transaction.

Ex :Consider Ram is transferring money to Shyam. The transaction must deduct the amount from the Ram's account and add it to the Shyam's account as a single operation.
If at any moment or at any part, this transaction fails (e.g., due to insufficient funds/system error/network error), the entire transaction is rolled back, ensuring that none of the accounts is affected partially.

LET'S START WITH DBMS :)

ACID Properties

C-> Consistency: It ensures that
1.**Read operations** retrieve consistent and up-to-date data from the database, and
2.**Write operations** ensure that data modifications maintain database constraints(such as foreign key relationships or unique constraints so that that data remains accurate)

It guarantees that the database remains in a consistent state before and after the execution of each transaction.

Ex: Consider you had 100rs in your account but you want 50rs cash, so you transferred 50rs to a person X and he gave you 50rs cash.
Before transaction- 100rs(in acc)
After transaction- 100rs(50rs in acc+ 50 rs cash)

LET'S START WITH DBMS :)

ACID Properties

I-> Isolation : It ensures that if there are two transactions 1 and 2, then the changes made by Transaction 1 are not visible to Transaction 2 until Transaction 1 commits.

While the transaction is reading data, the dbms ensures that the data is consistent and isolated from other transactions. This means that other transactions cannot modify the data being read by the current transaction until it is committed or rolled back.

Ex : A= 40rs (in DB)
Transaction 1 : Update value of A to 50rs
Transaction 2 : Read/get/Fetch value of A

If Transaction 1 is committed acc to Transaction 2 the value of A=50rs
If Transaction 1 is PENDING/RUNNING acc to Transaction 2 the value of A=40rs

LET'S START WITH DBMS :)

ACID Properties

D-> Durability : It ensures that once you save your data (commit a transaction), it stays saved, even if the system crashes or there is a power failure. Your data is always safe and won't disappear after you save as committed transactions are not lost.

Most DBMS use a technique called Write-Ahead Logging (WAL) to ensure durability. Before modifying data in the database, the DBMS writes the changes to a transaction log (often stored on disk) in a sequential manner. This ensures that if there is a failure event, the database can recover to a consistent state.

Ex : Consider if your are transferring 100rs to your friend and there is a sudden power outage or the system crashes right after the transaction is committed, the changes (the transfer of 100) will still be saved in the database. When the system is back up, both your account and your friend's account will reflect the updated balances.

LET'S START WITH DBMS :)

Isolation levels and its types

T1	T2
R	R
R	W
W	R
W	W

Why do we need to learn about isolation level?

In systems where multiple transactions are executed concurrently, isolation levels manage the extent to which the operations of one transaction are isolated from those of other transactions.

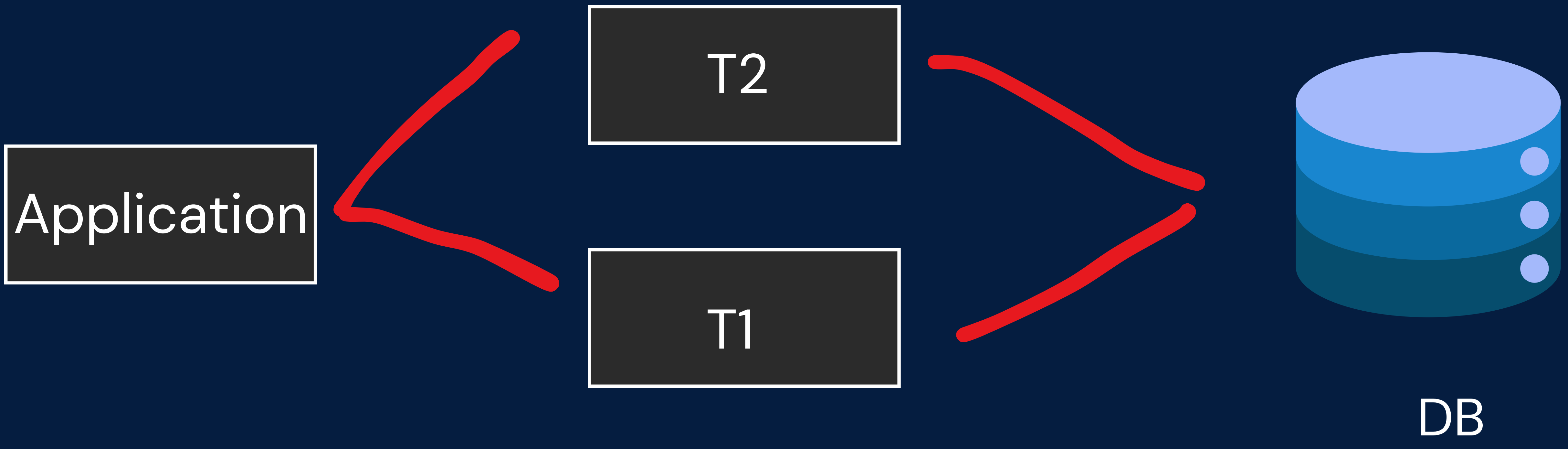
Isolation levels help prevent common transactional anomalies:

- Dirty Read: Reading uncommitted data from another transaction.
- Non-repeatable Read: Data changes after it has been read within the same transaction.
- Phantom Read: New rows are added or removed by another transaction after a query.

LET'S START WITH DBMS :)

<u>Isolation levels and its types</u>

Isolation level : It determines the degree to which the operations in one transaction are isolated from those in other transactions.



LET'S START WITH DBMS :)

Isolation levels and its types

Anamolies/Voilations to Isolation level

1. **Dirty Read :** Reading data written by a transaction that has not yet committed

Consider if T2 reads the data written by T1 and if T1 fails, it becomes irrelevant.

T1	T2
W(A)	
	R(A)

LET'S START WITH DBMS :)

Isolation levels and its types

Anamolies/Voilations to Isolation level

2. Non-Repeatable Read : Reading the same row twice within a transaction and getting different values because another transaction modified the row and committed.

Consider if T2 modifies the data which T1 already Read and if T1 continue the transaction the data will be changed

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

LET'S START WITH DBMS :)

Isolation levels and its types

Anamolies/Voilations to Isolation level

3. Phantom Read : Getting different sets of rows in subsequent queries within the same transaction because another transaction inserted or deleted rows and committed.

T1(Query(id))->Fetch the name
T2(Query(id)) -> Insert a new entry
T1(Query(id))->Fetch the name

LET'S START WITH DBMS :)

Isolation levels and its types

There are 4 isolation levels which helps us with these anamolies:

Types of Isolation Levels

- Read Uncommitted
- Read Committed
- Repeatable Read
- Serializable

LET'S START WITH DBMS :)

Isolation levels and its types

Read Uncommitted: The lowest isolation level where transactions can see uncommitted changes made by other transactions. If Transaction T1 is writing a value to a table, Transaction T2 can read this value before T1 commits.

- Dirty Reads: Yes
- Non-Repeatable Reads: Yes
- Phantom Reads: Yes

LET'S START WITH DBMS :)

Isolation levels and its types

Read Committed: It ensures that any data read during the transaction is committed at the moment it is read. If T1 has done some write operation T2 can only read the data when T1 is commited

- Dirty Reads: No
- Non-Repeatable Reads: Yes
- Phantom Reads: Yes

LET'S START WITH DBMS :)

Isolation levels and its types

Repeatable Read: It ensures that if a transaction reads a row, it will see the same values for that row during the entire transaction, even if other transactions modify the data and commit. If Transaction T1 reads a value, Transaction T2 cannot modify that value until T1 completes. But T2 can insert new rows that T1 can see on subsequent reads.

- Dirty Reads: No
- Non-Repeatable Reads: No
- Phantom Reads: Yes

LET'S START WITH DBMS :)

Isolation levels and its types

Serializable: It ensures a serial transaction execution, so that there is complete isolation. If Transaction T1 is executing, Transaction T2 must wait until T1 completes

- Dirty Reads: No
- Non-Repeatable Reads: No
- Phantom Reads: No

LET'S START WITH DBMS :)

Schedule and its Types

Schedule : It refers to the sequence in which a set of concurrent/multiple transactions are executed. You can also say it as a sequence in which the operations (such as read, write, commit, and abort) of multiple transactions are executed. It is really helpful to ensure data consistency and integrity.

If there are T1, T2, T3....TN (n) transactions then the possible schedules= n! (n factorial)

Ex : Schedule sc1

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

LET'S START WITH DBMS :)

Schedule and its Types

Incomplete schedule : An incomplete schedule is one where not all transactions have reached their final state of either commit or abort.

T1:Read(A)
T1:Write(A)
T2:Read(B)
T2:Write(B)
T2:COMMIT

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

Here, T1 is still in progress as there is no COMMIT for transaction T1.

LET'S START WITH DBMS :)

Schedule and its Types

Complete schedule : A complete schedule is one where all the transactions in the schedule have either committed or aborted.

T1:Read(A)
T1:Write(A)
T1:COMMIT
T2:Read(B)
T2:Write(B)
T2:COMMIT

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

LET'S START WITH DBMS :)

Schedule and its Types

Types of Schedule

1. Serial Schedule
2. Concurrent or Non-Serial Schedule
3. Conflict-Serializable Schedule
4. View-Serializable Schedule
5. Recoverable Schedule
6. Irrecoverable Schedule
7. Cascadeless Schedule
8. Cascading Schedule
9. Strict Schedule

LET'S START WITH DBMS :)

Schedule and its Types

1.Serial Schedule : A serial schedule is one where transactions are executed one after another. We can say it like if there are two transactions T1 and T2, T1 should commit to completeion before T2 starts.

Example : T1–>T2
T1:Read(A)
T1:Write(A)
T1:COMMIT(T1)
T2:Read(B)
T2:Write(B)
T2:COMMIT(T2)

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

T2 starts only after T1 is completed/ commited.

LET'S START WITH DBMS :)

Schedule and its Types

Serial Schedule

Advantages :

1. It follows the ACID properties. Transactions are isolated from each other.
2. It maintains consistency.

Challenges:

1. Since there is poor throughput(no of transactions completed per unit time) and memory utilisation, this is not suggested as it can be can be inefficient.
2. Since wait time is high, less no of transactions are completed.

LET'S START WITH DBMS :)

Schedule and its Types

2. Non-Serial/Concurrent Schedule : A non-serial schedule is one where multiple transactions can execute simultaneously(operations of multiple transactions are alternate/interleaved executions). We can say it like if there are two transactions T1 and T2, T2 doesn't need to wait for T1 to commit, it can start at any point.

Example : T1 ,T2, T3

T1	T2	T3
R(A)		
	R(A)	
		R(B)
	W(A)	
COMMIT		
		COMMIT

T2 didn't waited for T1 to commit

LET'S START WITH DBMS :)

Schedule and its Types

Non-Serial Schedule

Advantages :

1. Better resource utilization
2. Wait time is not involved. One transaction doesn't needs to wait for the running one to finish.
3. Throughput(no of transactions completed per unit time) is high

Challenges:

1. Consistentcy issue may arise because of non-serial execution. It requires robust concurrency control mechanisms to ensure data consistency and integrity.
2. We can use Serializability and Concurrency Control Mechanisms to ensure consistency.

LET'S START WITH DBMS :)

Schedule and its Types

3. Conflict-Serializable Schedule : A schedule is conflict-serializable if it can be transformed into a serial schedule by swapping adjacent non-conflicting operations.

R(A) T1 & R(A) T2: No conflict (both reads)
R(A) T1 & W(A) T2: RW conflict (T2 reads A after T1 writes A)
W(A) T1 & W(A) T2: WW conflict (both write to A)

Conflict pairs : Read-Write, Write-Read , Write-Write(perfomed on the same data item)
Non-conflict pairs :
Read-Read (perfomed on the same data item),
Read-Read (perfomed on the different data item),
Write-Write(perfomed on the different data item)

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

LET'S START WITH DBMS :)

Schedule and its Types

4. View-Serializable Schedule :

View serializability ensures that the database state seen by transactions in a concurrent schedule can be replicated by some serial execution of those transactions.

When we find cycle in our conflict graph, we don't know if our schedule is serializable or not, so we use the view serializability here.

A schedule is view serializable if it's view equivalent is equal to a serial schedule/execution.

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

LET'S START WITH DBMS :)

Schedule and its Types

5. Recoverable Schedule : A recoverable schedule ensures that if a transaction reads data from another transaction, it should not commit until the transaction from which it read has committed. This helps in maintaining the integrity of the database in case a transaction fails.

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
COMMIT	
	COMMIT

Recoverable Schedule

LET'S START WITH DBMS :)

Schedule and its Types

6. Irrecoverable Schedule : An irrecoverable schedule allows a transaction to commit even if it has read data from another uncommitted transaction. This can lead to inconsistencies and make it impossible to recover from certain failures.

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	COMMIT
FAIL	

Irrecoverable Schedule

LET'S START WITH DBMS :)

Schedule and its Types

7. Cascading Schedule : This schedule happens when the failure or abort of one transaction causes a series of other transactions to also abort.

- T1 Writes to A: T1 writes to data item A
- T2 Reads A: T2 reads the uncommitted value of A written by T1

Now, if T1 fails and aborts, T2 must also abort because it has read an uncommitted value from T1. Consequently, T3 must abort as well

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

Cascading Schedule

Issues:

1. Performance degradation because multiple transactions need to be rolled back
2. Improper CPU resource utilisation

LET'S START WITH DBMS :)

Schedule and its Types

8. Cascadeless Schedule : It ensures transactions only read committed data, such that the abort of one transaction does not lead to the abort of other transactions that have already read its uncommitted changes.

- T1 Writes to A: T1 writes to data item A
- T2 Reads A: T2 reads the committed value of A

Ensuring there is no write read problem(dirty read)
Also, T2 does not read any uncommitted data,
there are no cascading aborts in this schedule.

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

Cascadeless Schedule

Issues :

1. Write-Write problem is encountered.
2. Performance overhead is there

LET'S START WITH DBMS :)

Schedule and its Types

9. Strict Schedule : It ensures transaction is not allowed to read or write a data item that another transaction has written until the first transaction has either committed or aborted. It prevents cascading aborts.

- T2 cannot read or write the value of A until T1 has committed.
- This ensures that T2 only sees the committed value of A

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

Strict Schedule

LET'S START WITH DBMS :)

Concurrent VS Parallel Schedule

For	Concurrent	Parallel
Defination	Concurrent scheduling manages multiple tasks at the same time. Tasks can overlap in their execution periods but do not run exactly simultaneously.	Parallel scheduling runs multiple tasks simultaneously using multiple cores or processors. Each task is executed on a separate core or processor at the same time.
Execution	Achieved on a single-core processor through context switching, where the CPU rapidly alternates between tasks, creating the illusion of simultaneous execution.	Requires a multi-core processor or multiple processors, with each core/processor handling a different task concurrently.
Example	Multi-threading on a single-core CPU, where threads take turns using the CPU.	Multi-threading on a multi-core CPU, where threads run concurrently on different cores.

LET'S START WITH DBMS :)

Serializability and its types

Serializability: It ensures that concurrent transactions yield results that are consistent with some serial execution i.e the final state of the database after executing a set of transactions concurrently should be the same as if the transactions had been executed one after another in some order.

In case of concurrent schedule consistentcy issue may arise because of non-serial execution and we do serializability there, serial schedules are aready serial

LET'S START WITH DBMS :)

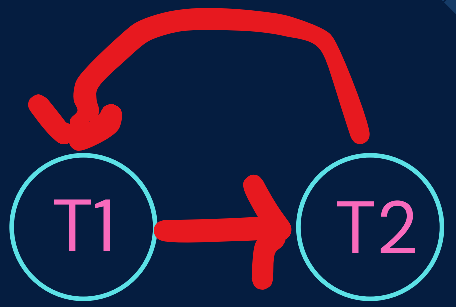
Consider nodes as transactions, and edges represent conflicts and detect if the resulting graph has cycles.

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

SERIAL SCHEDULE

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

Nodes: T1, T2
Edges : T1->T2, T2->T1
Cycle : Yes

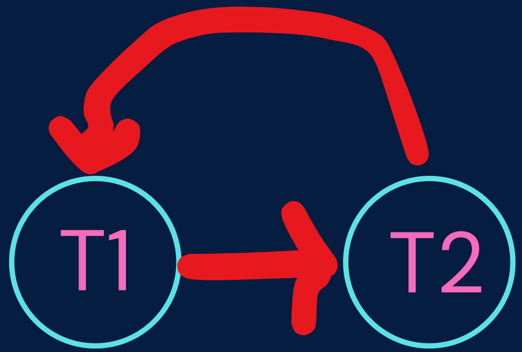

$$\begin{tikzcd} T1 \arrow[r] & T2 \arrow[l, bend right] \end{tikzcd}$$

CONCURRENT SCHEDULE

A concurrent schedule does not always have a cycle.
A concurrent schedule can be conflict-serializable, meaning that it is equivalent to some serial schedule of transactions and its conflict graph does not have any cycles.

LET'S START WITH DBMS :)

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


$$\begin{tikzcd} T1 \arrow[r] & T2 \arrow[l, bend right] \end{tikzcd}$$

CONCURRENT SCHEDULE

Nodes: T1, T2
Edges : T1->T2, T2->T1
Cycle : Yes

Now, since a cycle is detected we need to serialize them
So, we use the serializibilty here

- **Conflict-Serializability->** to detect the cycle using conflict graph
- **View-Serializability->** to check if schedule is serializable after a cycle is detected.

Why serializing them?

- To avoid consistentcy issue which may arise because of non-serial execution

LET'S START WITH DBMS :)

Serializability and its types

Types of Serializability

- Conflict-Serializability
- View-Serializability

LET'S START WITH DBMS :)

Conflict-Serializability

Serializibility : Serializability is a property of a schedule where the outcome is equivalent to some serial execution of the transactions.

Conflict-Serializability : A schedule is said to be conflict serializable when one of its conflict equivalent is serializabe. So basically, if a schedule can be transformed into a serial schedule by swapping non-conflicting operations, then the schedule is conflict serializable.

Conflict equivalent : If a schedule S1 is formed after swapping adjacent non-conflicting operations/pairs in a given schedule S, then S1 and S are conflict equivalent.

Conflict pairs : Read-Write, Write-Read , Write-Write(perfomed on the same data item)
Non-conflict pairs : Read-Read (perfomed on the same data item), Read-Read (perfomed on the different data item), Write-Write(perfomed on the different data item)

LET'S START WITH DBMS :)

Conflict-Serializability

Non- Conflict pairs

R(A)	
	R(B)

R(A)	
	R(A)

W(A)	
	W(B)

R(A)	
	W(B)

Conflict pairs

R(A)	
	W(A)

W(A)	
	R(A)

W(A)	
	W(A)

LET'S START WITH DBMS :)

Conflict-Serializability

How to achieve conflict equivalent schedule :

When a schedule can be transformed into a serial schedule by swapping adjacent non-conflicting operations. Conflicts arise when two transactions access the same data item, and at least one of them is a write operations.

Now, why are we only swapping the non-conflict pairs and not the conflict ones?

So if we swap the conflict pairs, the order of exceution if it was

T1 : R(A)

T2: W(A)

the results values may change as first we were reading A and then writing/modifying it, but now it will be writing A and then reading the modified value so the result might change if we change the order of execution.

LET'S START WITH DBMS :)

Conflict-Serializability

Q. Find a conflict equivalent for a schedule S1

S1

T1	T2
R(X)	
W(X)	
	R(X)
R(Y)	
W(Y)	
	R(Y)

LET'S START WITH DBMS :)

Conflict-Serializability

Q. Find a conflict equivalent for a schedule S1

1.Find the adjacent non-conflicting pairs and do a swap

T2 : R(X) T1: R(Y)

T1	T2
R(X)	
W(X)	
R(Y)	
	R(X)
W(Y)	
	R(Y)

T1	T2
R(X)	
W(X)	
	R(X)
R(Y)	
W(Y)	
	R(Y)

S1

LET'S START WITH DBMS :)

Conflict-Serializability

Q. Find a conflict equivalent for a schedule S1

2. After the first swap again search for adjacent non-conflicting pairs and swap if any
T2 : R(X) T1: W(Y)

T1	T2
R(X)	
W(X)	
R(Y)	
	R(X)
W(Y)	
	R(Y)

S1 after swap

T1	T2
R(X)	
W(X)	
R(Y)	
W(Y)	
	R(X)
	R(Y)

So, S1 is a serializable schedule
as S1' has a serial execution(i.e its conflict equivalent)

LET'S START WITH DBMS :)

When done on the same data item

- Read-Write (RW) Conflict
- Write-Read (WR) Conflict
- Write-Write (WW) Conflict

Conflict-Serializability

How to check whether a schedule is conflict-serializable or not?

Conflicts occur when two operations from different transactions access the same data item and at least one of them is a write operation.

Conflict graph/ precedence graph : A conflict graph, or precedence graph, is a directed graph used to determine conflict serializability. The nodes represent transactions, and the edges represent conflicts between transactions

LET'S START WITH DBMS :)

Conflict-Serializability

Conflict graph/ precedence graph:

- **Nodes:** Each transaction in the schedule is represented as a node in the graph.
- **Edges:** An edge from transaction T(X) to transaction T(Y)(denoted T(X)→T(Y)) is added if
  a. an operation of T(X) conflicts with an operation of T(Y) and
  b.T(X) operation precedes T(Y) operation in the schedule.
- **Cycle Detection:** The schedule is conflict-serializable if and only if the conflict graph is acyclic. If there are no cycles in the graph, it means that the schedule can be serialized without violating the order of conflicting operations.

LET'S START WITH DBMS :)

Conflict-Serializability

Q. Check if the given schedule is conflict-serializable or not?

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

- T1 reads A
- T2 reads A
- T1 writes A
- T3 writes A
- T2 writes B
- T3 reads B

LET'S START WITH DBMS :)

Conflict-Serializability

Q. Check if the given schedule is conflict-serializable or not?

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

Step 1: Find the conflict in the schedule

- RW Conflict (T1, T3) on A
- RW Conflict (T2, T3) on A
- RW Conflict (T2.T1) on A
- WW Conflict (T1, T3) on A
- WR Conflict(T2.T3) on B

Step 2: Find the nodes

Each transaction T1 T2, T3 is a node in the graph

Step 3: Find the edges/conflicts

- T1→T3: Because T1 reads A before T3 writes A.
- T2→T3: Because T2 reads A before T3 writes A
- T1→T3: Because T1 writes A before T3 writes A.
- T2→T3: Because T2 writes B before T3 reads B.
- T2->T1 : Because T2 reads A before T1 writes A.

Write-Write conflict	Write-Read conflict
W(B) - W(B)	W(B) - R(B)
W(A)-W(A)	W(A)-R(A)

Read-write conflict
R(B) - W(B)
R(A)-W(A)

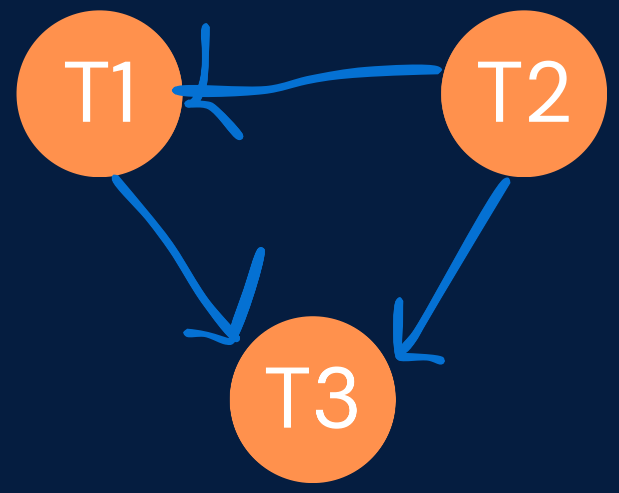
LET'S START WITH DBMS :)

Conflict-Serializability

Q. Check if the given schedule is conflict-serializable or not?

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

Step 4 : Draw the graph


$$\begin{tikzcd} T1 \arrow[dr] & & T2 \arrow[ll] \arrow[dl] \\ & T3 & \end{tikzcd}$$

edges

- T2→T1
- T1→T3
- T2→T3

Step 5 : If the graph has cycle it is not confliict-serializable or serializable, if not lets find the serial execution of transactions

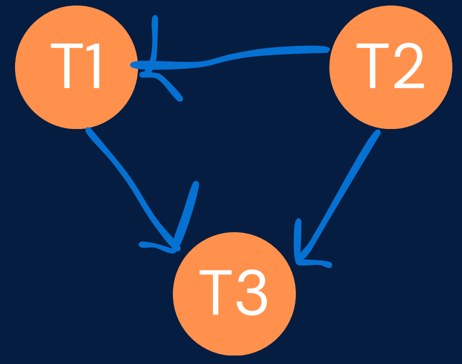
Since the conflict graph is acyclic, the schedule is conflict-serializable

LET'S START WITH DBMS :)

Conflict-Serializability

Q. Check if the given schedule is conflict-serializable or not?

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



edges
T2→T1
T1→T3
T2→T3

Step 6 : Lets find the serial execution of transactions

Possible combinations are :

T1->T2->T3
T1->T3->T2
T2->T1->T3
T2->T3->T1
T3->T2->T1
T3->T1->T2

Find the indegree(the number of edges directed into that node) and if its 0 it can be the first in serial execution

T1 - 1 ,T2- 0, T3- 2 , T2 would be the first as indegree is 0

- T2 must precede T1
- T1 must precede T3

Therefore, one possible equivalent serial schedule is **T2→T1→T3.**

LET'S START WITH DBMS :)

View-Serializability

View serializability ensures that the database state seen by transactions in a concurrent schedule can be replicated by some serial execution of those transactions.

When we find cycle in our conflict graph, we don't know if our schedule is serializable or not, so we use the view serializability here.

A schedule is view serializable if it's view equivalent is equal to a serial schedule/execution.

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

LET'S START WITH DBMS :)

View-Serializability

Conditions for View Equivalent schedule
For a schedule to be view-Equivalent it should follow the following conditions

1.Initial Read Values:

- An initial read of both schedules must be identical. For example, consider two schedules, S and S'. If, in schedule S, transaction T1 reads the data item A, then in schedule S', transaction T1 should also read A.

S

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

S'

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

LET'S START WITH DBMS :)

View-Serializability

Conditions for View Equivalent schedule

2.Updated Read:

- In schedule S, if Ti is reading B which is updated by Tj then in S' also, Ti should read B which is updated by Tj.

S

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

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

S'

LET'S START WITH DBMS :)

View-Serializability

Conditions for View Equivalent schedule

3. Final Update

- A final write must be identical in both schedules. If, in schedule S, transaction Ti is the last to update A, then in schedule S', the final write operation on A should also be performed by Ti

S

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

S'

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

LET'S START WITH DBMS :)

View-Serializability

The number of possible serial schedules for n transactions is given by the number of permutations of the transactions: n!

Q.Find the view equivalent schedule

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

LET'S START WITH DBMS :)

View-Serializability

Step 1: Find if conflict serializable or not.
Step 2: Find the possible serial schedules -> 3!
Step 3: Choose one possibility and check for view equivalent conditions (T1->T2->T3)

1. Initial Read -> T1
2. Update Read-> No read performed after update so no need to check
3. Final Update-> T3

Step 4: If the view equivalent schedule matches the condition, it is view serializable.

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

LET'S START WITH DBMS :)

Concurrency control mechanisms

Concurrency control -> It is a process of mananging multiple users access and modification in data simultaneously in shared or multi-user database systems.

How it helps?

1. **Data Consistency**: Ensures that data remains accurate and reliable despite concurrent access.
2. **Isolation**: Maintains the isolation property of transactions, so the outcome of a transaction is not affected by other concurrently executing transactions.
3. **Serializability**: Ensures that the result of concurrent transactions is the same as if the transactions had been executed serially

LET'S START WITH DBMS :)

Concurrency control mechanisms

Concurrency control mechanisms are techniques used in DBMS to ensure that transactions are executed concurrently without leading to inconsistencies in the database. In serializability we check if a schedule is serializable or not but in concurrency control we use certain techniques to make a schedule serializable.

Why its needed?

- **Lost Updates:** When two or more transactions update the same data simultaneously, one of the updates might be lost. For example, if two users modify the same record at the same time, the changes made by one user could overwrite the changes made by the other. (WW)

- **Dirty Reads:** When a transaction reads data that has been modified by another transaction but not yet committed. If the first transaction rolls back, the other transaction will have read invalid data. (WR)

LET'S START WITH DBMS :)

Concurrency control mechanisms

- **Uncommitted Dependency (Dirty Writes):** When a transaction modifies data that has been read by another transaction, leading to inconsistencies if one of the transactions rolls back.

- **Inconsistent Retrievals (Non-repeatable Reads):** Happens when a transaction reads the same data multiple times and gets different values each time because another transaction is modifying the data in between the reads.

- **Phantom Reads:** Occurs when a transaction reads a set of rows that satisfy a condition, but another transaction inserts or deletes rows that affect the set before the first transaction completes. This results in the first transaction reading different sets of rows if it re-executes the query.

LET'S START WITH DBMS :)

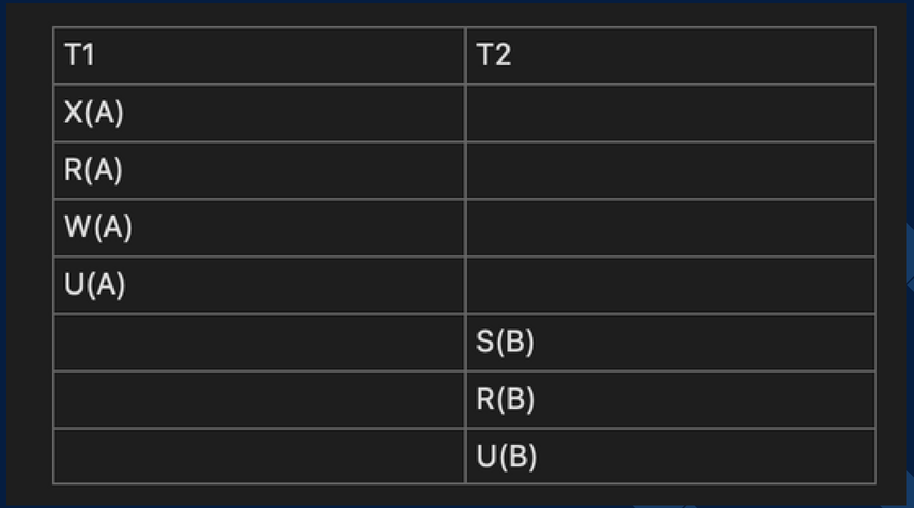
Concurrency control mechanisms

Some common concurrency control mechanisms are :

- **Lock-Based Protocols**(2PL, Strict 2PL, Rigorous 2PL , Conservative 2PL)
- **Timestamp-Based Protocols** (Basic Timestamp Ordering (TO))
- **Optimistic Concurrency Control** (OCC)
- **Multiversion Concurrency Control** (MVCC)

LET'S START WITH DBMS :)

Concurrency control mechanisms



T1	T2
X(A)	
R(A)	
W(A)	
U(A)	
	S(B)
	R(B)
	U(B)

- Lock-Based Protocols

Types of Locks

a. **Binary Locks**: A simple mechanism where a data item can be either locked (in use) or unlocked.If a thread tries to acquire the lock when it's already locked, it must wait until the lock is released by the thread currently holding it.

b. **Shared and Exclusive Locks**

- **Shared Lock (S-lock):** Allows multiple transactions to read a data item simultaneously but prevents any of them from modifying it. Multiple transactions can hold a shared lock on the same data item at the same time.
- **Exclusive Lock (X-lock):** Allows a transaction to both read and modify a data item. When an exclusive lock is held by a transaction, no other transaction can read or modify the data item.

LET'S START WITH DBMS :)

Concurrency control mechanisms

Note : When a transaction acquires a shared lock on a data item, other transactions can also acquire shared locks on that same item, enabling concurrent reads. However, no transaction can acquire an exclusive lock on that item as long as one or more shared locks are held.

When a transaction acquires an exclusive lock on a data item, it has full control over that item, meaning it can both read and modify it. No other transaction can acquire a lock on the same data item until the exclusive lock is released.

Shared lock -> any no of transactions
Exclusive lock -> only one transaction should hold it at one time

While shared and exclusive locks are vital for maintaining data integrity and consistency in concurrent environments, they can introduce significant challenges in terms of performance, deadlocks, reduced concurrency, and system complexity.

LET'S START WITH DBMS :)

Concurrency control mechanisms

Drawbacks of shared-exclusive locks

Performance issues : Managing locks requires additional CPU and memory resources. The process of acquiring, releasing, and managing locks can introduce significant overhead

Concurrency issues : Exclusive locks prevent other transactions from accessing locked data, which can significantly reduce concurrency.

Starvation: Some transactions may be delayed if higher-priority transactions consistently acquire locks before them, leading to starvation where a transaction never gets to proceed.

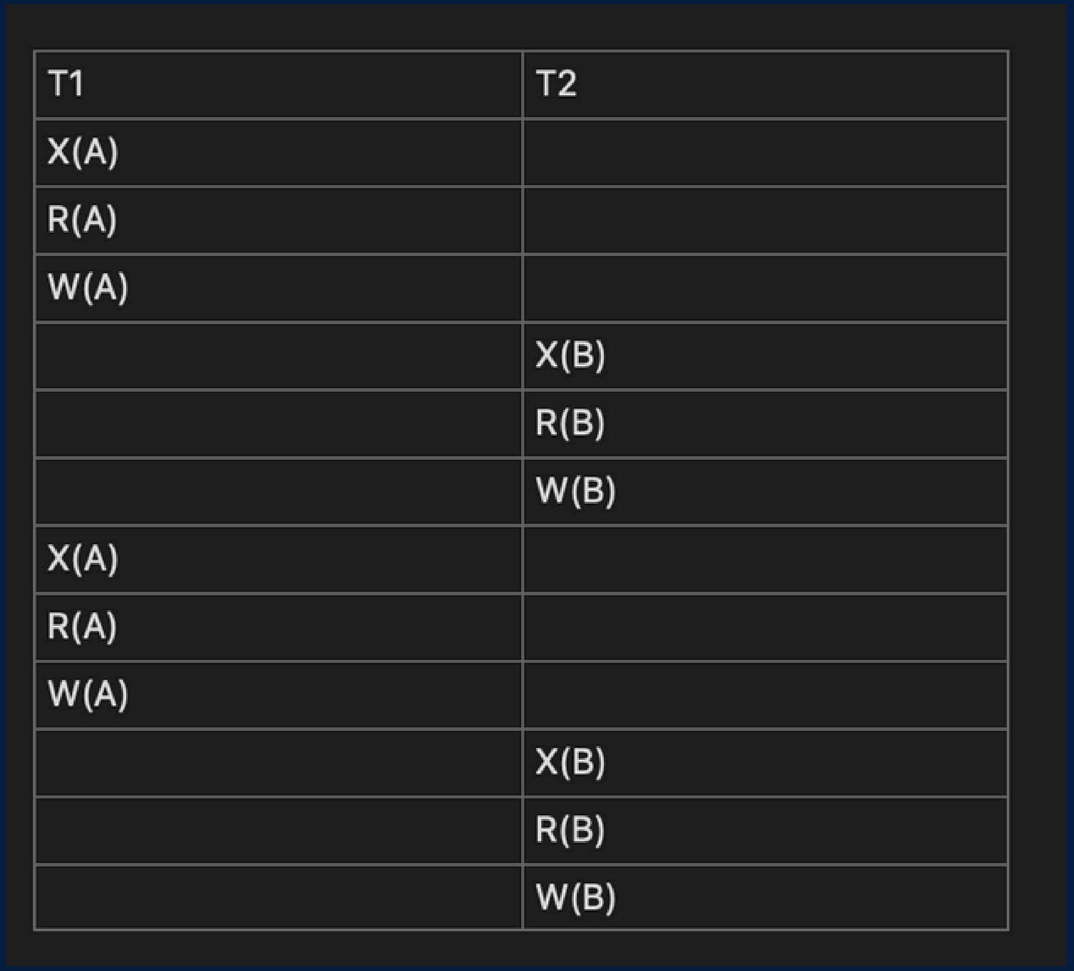
Deadlocks : Shared and exclusive locks can lead to deadlocks, where two or more transactions hold locks that the other transactions need.

Irrecoverable : If Transaction B commits after the lock is release based on a modified value in transaction A which fails after sometime.

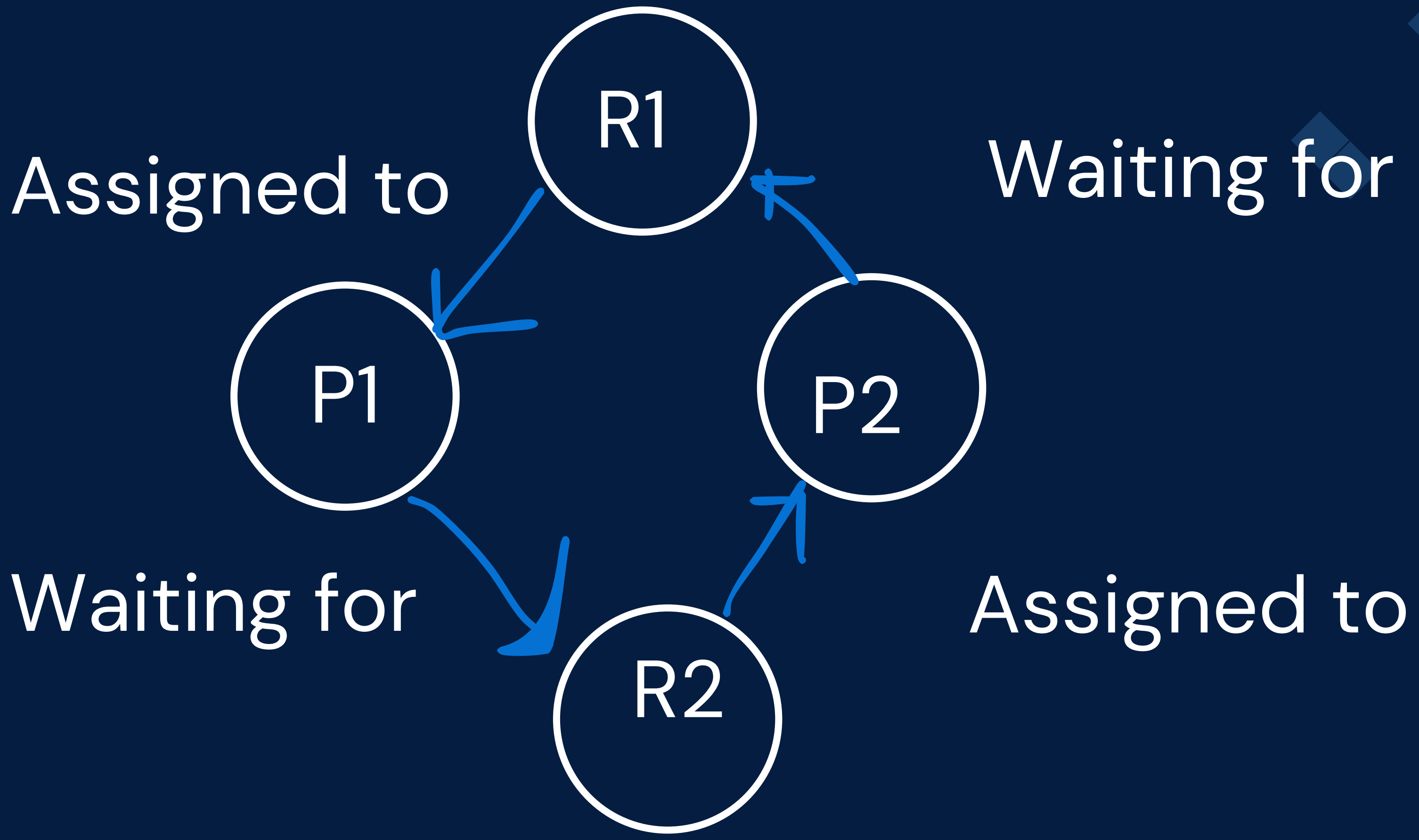
LET'S START WITH DBMS :)

Concurrency control mechanisms

Deadlock : It is a situtaion when 2 or more transactions wait for one another to give up the locks.



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



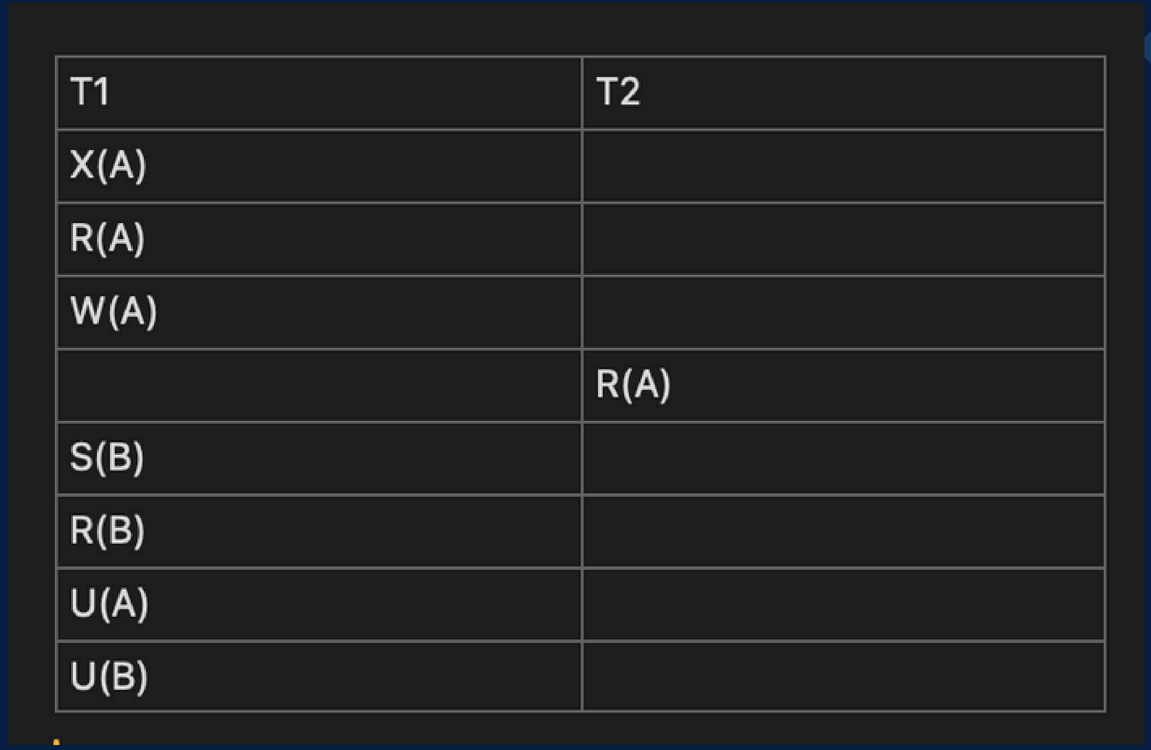
LET'S START WITH DBMS :)

Concurrency control mechanisms

Two-Phase Locking (2PL) : This protocol ensures serializability by dividing the execution of a transaction into two distinct phases

- **Growing Phase:** A transaction can acquire locks but cannot release any. This phase continues until the transaction has obtained all the locks it needs.

- **Shrinking Phase:** After the transaction releases its first lock, it can no longer acquire any new locks. During this phase, the transaction releases all the locks it holds.



T1	T2
X(A)	
R(A)	
W(A)	
	R(A)
S(B)	
R(B)	
U(A)	
U(B)	

Any transaction which is following 2PL locking achieves serializability and consistency.

LET'S START WITH DBMS :)

Concurrency control mechanisms

Two-Phase Locking (2PL)

Advantages :

1. It guarantees that the schedule of transactions will be serializable, meaning the results of executing transactions concurrently will be the same as if they were executed in some serial order.
2. By ensuring that transactions are serializable, 2PL helps maintain data integrity and consistency, which is critical in environments where data accuracy is essential.

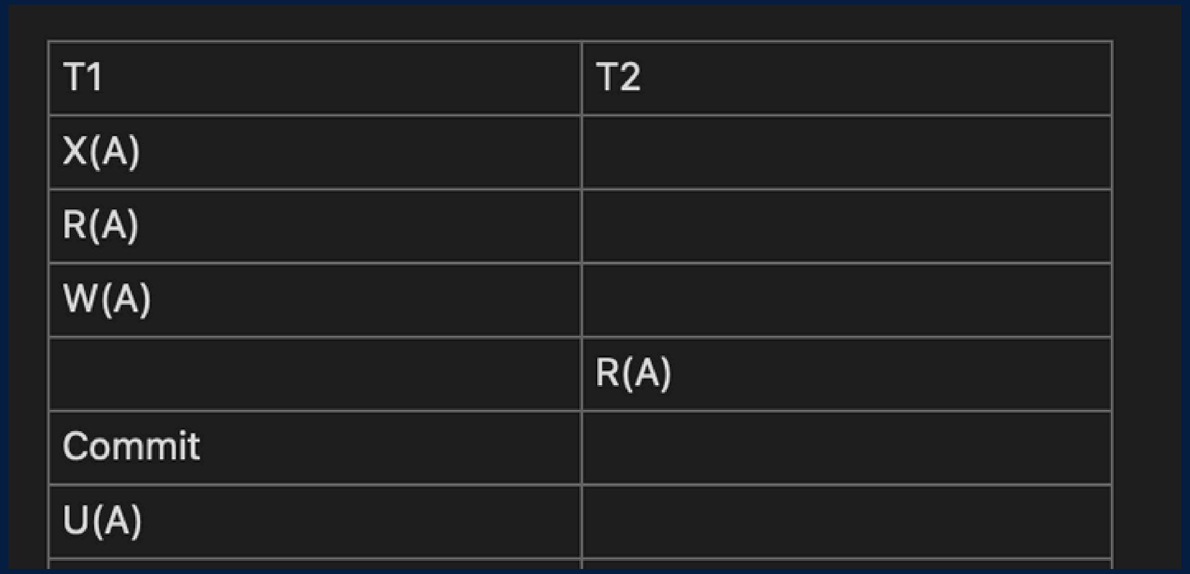
Disadvantages :

1. Deadlocks, starvation and cascading rollbacks
2. Transactions must wait for locks to be released by other transactions. This can lead to increased waiting times and lower system throughput.
3. In case of a system failure, recovering from a crash can be complex

LET'S START WITH DBMS :)

Concurrency control mechanisms

Strict Two-Phase Locking (Strict 2PL):
A stricter variant where exclusive locks are held until the transaction commits or aborts. This helps prevent cascading rollbacks (where one transaction's rollback causes other transactions to roll back).



T1	T2
X(A)	
R(A)	
W(A)	
	R(A)
Commit	
U(A)	

Advantages:

- Prevents Cascading Aborts
- Ensures Strict Serializability

Disadvantages:

- Since write locks are held until the end of the transaction, other transactions may be blocked for extended periods
- Transactions may experience longer wait times to acquire locks
- Deadlocks and starvation is there

LET'S START WITH DBMS :)

Concurrency control mechanisms

Rigorous Two-Phase Locking:
An even stricter version where all locks (both shared and exclusive) are held until the transaction commits. This guarantees strict serializability.

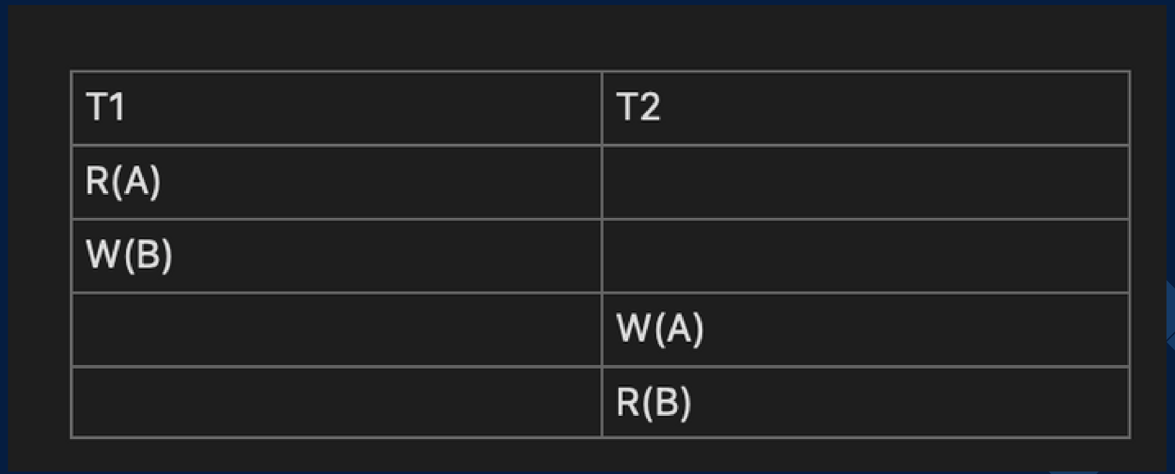
Advantages:

- Since all locks are held until the end of the transaction, the system can easily ensure that transactions are serializable and can be recovered
- Prevents Cascading Aborts and Dirty Reads
-

Disadvantages:

- Performance bottlenecks
- Increased Transaction Duration
- Deadlocks and starvation is there

LET'S START WITH DBMS :)



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

Concurrency control mechanisms

Conservative Two-Phase Locking:

Conservative Two-Phase Locking(Static Two-Phase Locking) is a variant of the standard 2PL protocol that aims to prevent deadlocks entirely by requiring a transaction to acquire all the locks it needs before it begins execution.

If the transaction is unable to acquire all the required locks (because some are already held by other transactions), it waits and retries. The transaction only starts execution once it has successfully acquired all the necessary locks.

Since a transaction never starts executing until it has all the locks it needs, deadlocks cannot occur because no transaction will ever hold some locks and wait for others

In this scenario, deadlocks cannot occur because neither T1 nor T2 starts execution until it has all the locks it needs.

LET'S START WITH DBMS :)

Concurrency control mechanisms

Timestamp-Based Protocols: It assign a unique timestamp

- Every transaction is assigned a unique timestamp when it enters the system. It is used to order the transactions based on their Timestamps.

- There are two important timestamps for each data item:
    - **Read Timestamp (RTS):** The last timestamp of any transaction that has successfully read the data item.

    - **Write Timestamp (WTS):** The last timestamp of any transaction that has successfully written the data item.

Transaction with smaller timestamp(Old) -> Transaction with larger timestamp(young)

LET'S START WITH DBMS :)

Concurrency control mechanisms

Timestamp-Based Protocols: It assign a unique timestamp

Rules : Consider if there are transactions performed on data item A.

R_TS(A) -> Read time-stamp of data-item A.

W_TS(A)-> Write time-stamp of data-item A.

1.Check the following condition whenever a transaction Ti issues a Read (X) operation:

- If W_TS(A) >TS(Ti) then the operation is rejected. (rollback Ti)
- If W_TS(A) <= TS(Ti) then the operation is executed. (set R_TS(A) as the

max of (R_TS(A), TS(Ti)

2. Check the following condition whenever a transaction Ti issues a Write(X) operation:

- If TS(Ti) < R_TS(A) then the operation is rejected. (rollback Ti)
- If TS(Ti) < W_TS(A) then the operation is rejected and Ti is rolled back otherwise the operation is executed. Set W_TS(A)=TS(Ti)

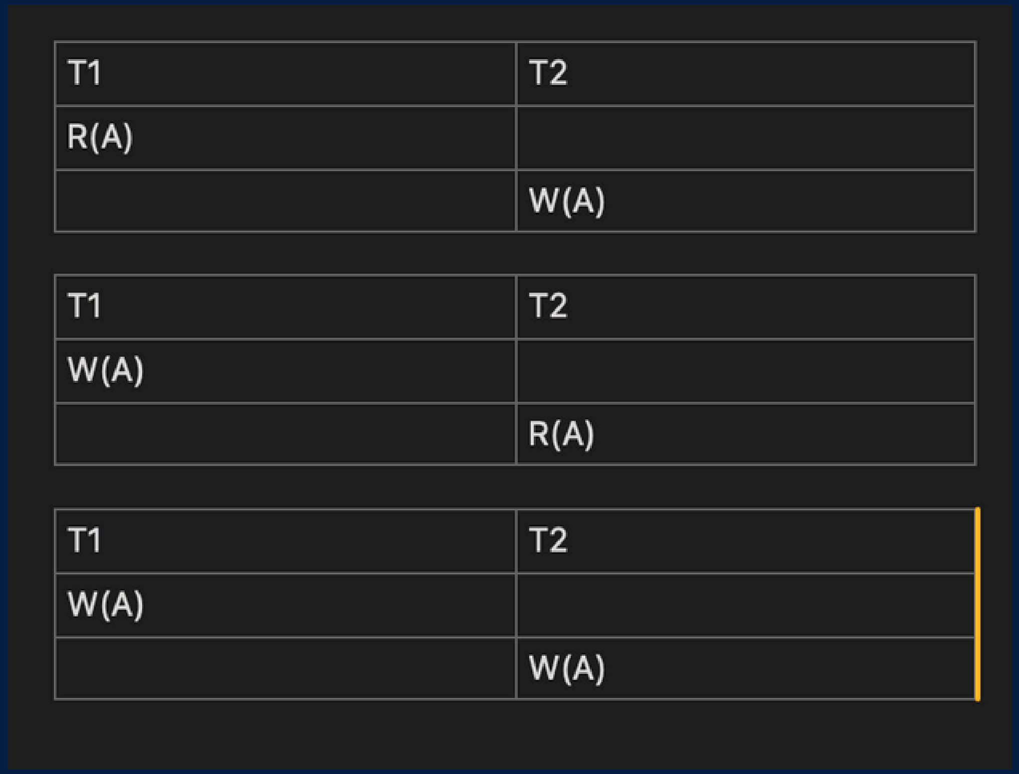
LET'S START WITH DBMS :)

Concurrency control mechanisms

Timestamp-Based Protocols: It assign a unique timestamp

Let's assume there are two transactions T1 and T2. Suppose the transaction T1 has entered the system at 7:00PM and transaction T2 has entered the system at 7:05pm then T1 has the higher priority, so it executes first as it is entered the system first. T1->T2

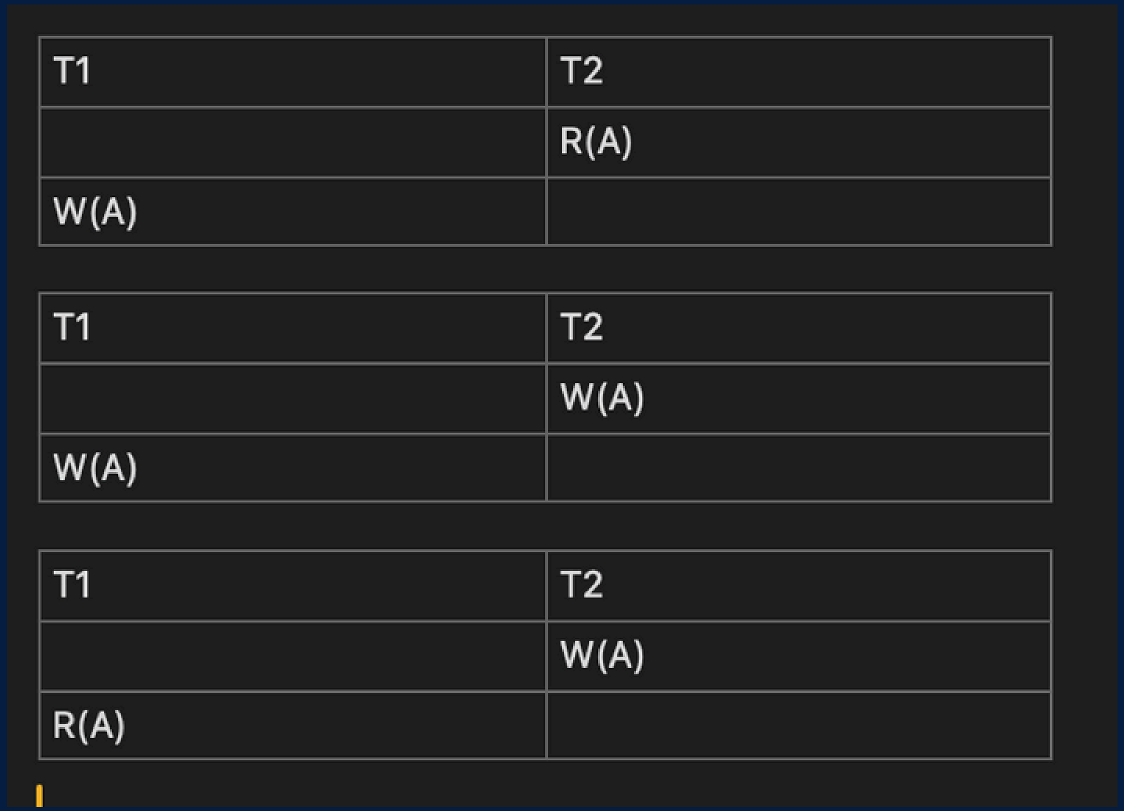
If younger has done a Read/Write and older wants to do Read/Write, thats a rollback case.



T1	T2
R(A)	
	W(A)

T1	T2
W(A)	
	R(A)

T1	T2
W(A)	
	W(A)



T1	T2
	R(A)
W(A)	

T1	T2
	W(A)
W(A)	

T1	T2
	W(A)
R(A)	

LET'S START WITH DBMS :)

Database recovery management

It involves strategies and processes to restore a database to a consistent state after a failure or crash.

Types of Database Failures

- **Transaction Failure:** Occurs when a transaction cannot complete successfully due to logical errors or system issues (like deadlocks).
- **System Failure:** Occurs when the entire system crashes due to hardware or software failures, leading to loss of in-memory data.
- **Media Failure:** Occurs when the physical storage (e.g., hard drives) is damaged, resulting in data loss or corruption.

LET'S START WITH DBMS :)

Database recovery management

Recovery Phases

- **Analysis Phase:** Identifies the point of failure and the transactions that were active at that time.

- **Redo Phase:** Reapplies changes from committed transactions to ensure the database reflects all completed operations.

- **Undo Phase:** Reverts the effects of incomplete transactions to maintain consistency.

LET'S START WITH DBMS :)

Database recovery management

Recovery Techniques

- **Backup and Restore**: Regular backups are taken to ensure data can be restored. Full, incremental, and differential backups are common types.

- **Logging**: Keeps a record of all transactions. The Write-Ahead Logging (WAL) protocol ensures that logs are written before any changes are applied to the database.

- **Shadow Paging**: Maintains two copies of the database pages; one is updated, and the other remains unchanged until the transaction commits.