[1] Introduction to Database Transaction

- Database Transaction is an atomic unit that contains one or more SQL statements.
- It is a series of operations that performs as a single unit of work against a database.
- It is a logical unit of work.
- It has a beginning and an end to specify its boundary.

Let's take an simple example of bank transaction, Suppose a Bank clerk transfers Rs. 1000 from X's account to Y's account.

X's Account

```
open-account (X)
prev-balance = X.balance
curr-balance = prev-balance – 1000
X.balance = curr-balance
close-account (X)
```

Decreasing Rs. 1000 from X's account, saving new balance that is current balance and after completion of transaction the last step is closing the account.

Y's Account

```
open-account (Y)
prev - balance = Y.balance
curr - balance = prev-balance + 1000
Y.balance = curr-balance
close-account (Y)
```

Adding Rs. 1000 in the Y's account and saving new balance that is current balance and after completion of transaction the last step is closing the account.

- The above example defines a very simple and small transaction that tells how the transaction management actual works.

What is a transaction?

"Transaction is a set of operations which are all logically related."

OR

"Transaction is a single logical unit of work formed by a set of operations."

Operations in a transaction-

The main operations in a transaction are-

1. Read Operation
2. Write Operation

1. Read Operation-

Read operation reads the data from the database and then stores it in the buffer of main memory.

- For example- **Read(A)** instruction will read the value of A from the database and will store it in the buffer of main memory.

2. Write Operation-

Write operation writes the data back to the database from the buffer. For example-

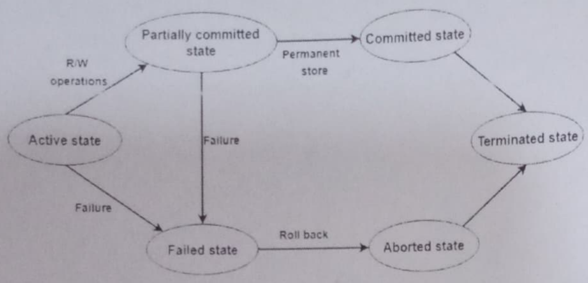
- **Write(A)** will write the updated value of A from the buffer to the database.

Transaction States-

A transaction goes through many different states throughout its lifetime. These states are known as **transaction states**.

Transaction states are as follows-

1. **Active state**
2. **Partially committed state**
3. **Committed state**
4. **Failed state**
5. **Aborted state**
6. **Terminated state**



1. Active state-

This is the first state in the lifecycle of a transaction.

- A transaction is called in an **active state** as long as its instructions are getting executed from first to last.
- All the changes made by the transaction are being stored in a buffer in main memory.

2. Partially committed state-

After the last instruction of the transaction gets executed, it enters into a **partially committed state**.

- After the transaction has entered into this state, the transaction is now considered to be partially committ
- It is not considered to be fully committed because all the changes made by the transaction are still stored only in the buffer in main memory and not into the database.

3. Committed state-

- After all the changes made by the transaction have been successfully stored into the database, the transac enters into a **committed state**.

NOTE-

- After a transaction has entered the committed state, it is not possible to roll back the transaction i.e. we c not undo the changes the transaction has made because the system has been now updated into a new consistent state.
- The only best possible thing that can be done to undo the changes made by the transaction is to carry out another transaction called **compensating transaction** to reverse the changes.

4. Failed state-

- When a transaction is present in the active state or partially committed state i.e. when a transaction is be executed or has partially committed and and some failure occurs due to some reason and it is analyzed th the normal execution is now impossible, the transaction then enters into a **failed state**.

5. Aborted state-

- After the transaction has failed and has entered into a failed state, all the changes made by the transaction have to be undone.
- To undo the changes made by the transaction, it is necessary to roll back the transaction.
- After the transaction has rolled back completely, it enters into an **aborted state.**

6. Terminated state-

This is the last state in the life cycle of a transaction.

- After entering the committed state or aborted state, the transaction then finally enters into a **terminated state** where the transaction life cycle finally comes to an end.

[1.1] Transaction Properties

Following are the Transaction Properties, referred to by an acronym ACID properties:

1. Atomicity
2. Consistency
3. Isolation
4. Durability

- ACID properties are the most important concepts of database theory.
- A transaction is a small unit of program which contains several low level tasks.
- These properties guarantee that the database transactions are processed reliably.

1. Atomicity

- Atomicity defines that all operations of the transactions are either executed or none.
- Atomicity is also known as 'All or Nothing', it means that either perform the operations or not perform at all.
- It is maintained in the presence of deadlocks, CPU failures, disk failures, database and application software failures.
- It can be turned off at system level and session level.
- Atomicity control by **transaction management system**

2. Consistency

- Consistency defines that after the transaction is finished, the database must remain in a consistent state.
- It preserves consistency of the database.
- If execution of transaction is successful, then the database remains in a consistent state. If the transaction fails, then the transaction will be rolled back and the database will be restored to a state consistent.

  Consistency control by Atomicity, Isolation Durability

3. Isolation

- Isolation defines that the transactions are securely and independently processed at the same time without interference.
- Isolation property does not ensure the order of transactions.
- The operations cannot access or see the data in an intermediate state during a transaction.
- Isolation is needed when there are concurrent transactions occurring at the same time.
- **Isolation control by concurrency control component**

4. Durability

- Durability states that after completion of transaction successfully, the changes are required for the database.
- Durability holds its latest updates even if the system fails or restarts.
- It has the ability to recover committed transaction updates even if the storage media fails.
- **Durability control by recovery management component**

[1.3] Schedules

The order in which the operations of multiple transactions executing concurrently appears for execution is known as a schedule. In a database system, we can have number of transaction processing. Related transactions will be processed one after another. There are some transactions processes in parallel. Some of the transactions can be grouped together.

A **schedule** is a process of grouping the transactions into one and executing them in a predefined order schedule is required in a database because when some transactions execute in parallel, they may affect result of the transaction – means if one transaction is updating the values which the other transaction accessing, then the order of these two transactions will change the result of second transaction. Hence schedule is created to execute the transactions. A schedule is called **serial schedule**, if the transactions in schedule are defined to execute one after the other.

Even when we are scheduling the transactions, we can have two transactions in parallel, if they independent. But if they are dependent by any chance, then the results will change. For example, say transaction is updating the marks of a student in one subject; meanwhile another transaction is calculating total marks of a same student. If the second transaction is executed after first transaction is complete, then the transaction will be correct. But what if second transaction runs first? It will have wrong total

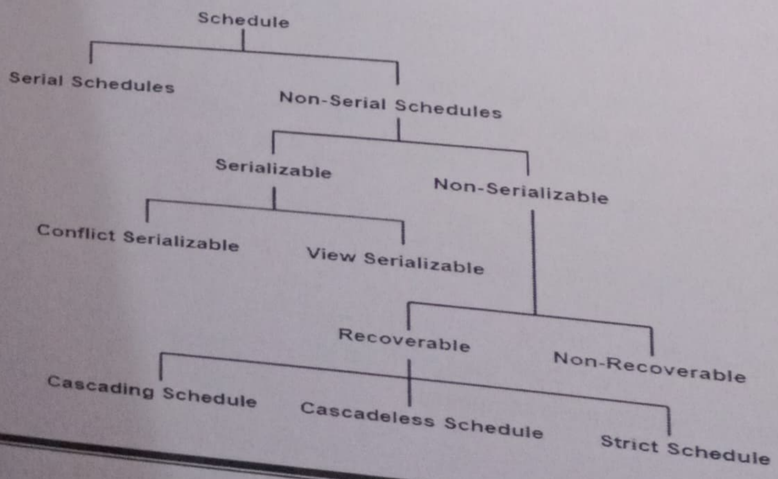
This type of incorrect processing of transaction needs to be handled in the system. Parallel executio transaction is allowed in the database only if there is any equivalence relation among the transactions. T are three types of equivalence relation among the transactions – Result, view and Conflict Equivalence.

Result Equivalence: - If the two transactions generate same result after their execution then it is calle result equivalence. For example, one transaction is updating the marks of Student X and the other transac is to insert a new student. Here both the transactions are totally independent and their order of execution not matter. Whichever order they are executed; the result is the same. Hence it is called result equivale transactions.

View Equivalence: -Two schedules are said to be view equivalence, if the transaction in one schedule is s as the one in other. That means, both the transactions in two schedules perform same tasks. For example, schedule1 has transaction to insert new students into STUDENT table and second schedule has the transac to maintain the old student records in a new table called OLD_STUDENT. In both the schedules stu details are inserted, but different tables (it does not matter if it is same table). Such schedules are calle view equivalence schedule.

Conflict Equivalence: - In this case both schedules will have different set of transactions, but they woul accessing same data and one of the schedules will be inserting/updating the records. In this equivalence, the schedules will have conflicting set of transactions. Above example of updating the marks of one stude one transaction and calculating the total marks at the same time is a conflicting equivalence. In all these cases if the transaction is serialized, then the issues can be resolved.

<u>Classification of schedules-</u>



Serial Schedules-

Serial schedules are those schedules in which all the transactions execute one by one i.e. when the execution of one transaction is in process, no other transaction executes.

Serial schedules are always-

- Consistent
- Recoverable
- Cascadeless
- Strict

Example-01:

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

In this schedule, we have two transactions T1 and T2 where transaction T1 executes first and after it complete its execution, transaction T2 begins its execution. So, this schedule is an example of a **serial schedule**.

Example-02:

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

In this schedule also, we have two transactions T1 and T2 where transaction T2 executes first and after it complete its execution, transaction T1 begins its execution. So, this schedule also is an example of a **serial schedule**.

Non-Serial Schedules-

Non-serial schedules are those schedules in which the operations of all transactions are interleaved or mixed with each other.

Non-Serial schedules are **NOT** always-

- Consistent
- Recoverable
- Cascadeless
- Strict

Example-01:

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

In this schedule, we have two transactions T1 and T2 where the operations of T1 and T2 are interleaved. So, this schedule is an example of a **non-serial schedule**

Example-02:

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

In this schedule also, we have two transactions T1 and T2 where the operations of T1 and T2 are interleaved. So, this sched
also is an example of a **non-serial schedule**.

Finding the possible number of schedules-

Suppose we have 'n' number of transactions T1, T2, T3 …. , Tn with N1, N2, N3 …. , Nn number of operations respective

Then- **Total number of schedules (Serial + Non-Serial) is given by-**

$$\boxed{\frac{N1 + N2 + N3 + \ldots\ldots + Nn}{N1! \times N2! \times N3! \times \ldots\ldots \times Nn!}}$$

Total number of serial schedules is given by-

n!

Total number of non-serial schedules is given by-

Total Number of Schedules – Total Number of Serial Schedules

PRACTICE PROBLEM BASED ON FINDING THE NUMBER OF SCHEDULES-

Problem-

Consider any three transactions with 2, 3, 4 operations respectively, find-

1. How many total number of schedules are possible?
2. How many total number of serial schedules are possible?
3. How many total number of non-serial schedules are possible?

Solution-

Let us solve each part one by one-

Total number of schedules (Serial + Non-Serial) that are possible is given by-

$$= \frac{(2+3+4)!}{2! \times 3! \times 4!}$$

$$= 1260$$

Total number of serial schedules that are possible is given by-

3! = 6

Total number of non-serial schedules that are possible is given by-

= Total number of schedules possible – Total number of serial schedules

= 1260 – 6

= 1254

[2] What is serializability?

- When multiple transactions run concurrently, then it may give rise 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.
- Serializability is a concurrency scheme where the concurrent transaction is equivalent to one that executes the transactions serially.
- A schedule is a list of transactions.
- Serial schedule defines each transaction is executed consecutively without any interference from other transactions.
- Non-serial schedule defines the operations from a group of concurrent transactions that are interleaved.
- In non-serial schedule, if the schedule is not proper, then the problems can arise like multiple update, uncommitted dependency and incorrect analysis.
- The main objective of serializability is to find non-serial schedules that allow transactions to execute concurrently without interference and produce a database state that could be produced by a serial execution.

Serializable Schedules-

- If a given schedule of 'n' transactions is found to be equivalent to some serial schedule of 'n' transactions, then it is called as a **serializable schedule**.

Properties of Serializable Schedules-

Serializable schedules behave exactly same as serial schedules. So, they are always-

- Consistent

- Recoverable
- Casacadeless
- Strict

Difference between Serial Schedules and Serializable Schedules-

The only difference between serial schedules and serializable schedules is that-

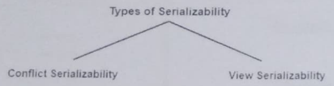
- In serial schedules, only one transaction is allowed to execute at a time i.e. no concurrency is allowed.
- Whereas in serializable schedules, multiple transactions can execute simultaneously i.e. concurrency is allowed

Which is better- Serial Schedules or Serializable schedules?

We have mentioned that serial schedules and serializable schedules behave exactly same. The added advantage with ser schedules is that they allow multiple transactions to execute simultaneously i.e. they allow concurrency.

Since, serializable schedules improve both resource utilization and CPU throughput, therefore **serializable schedules ar always better than serial schedules.**

Types of Serializability-



Serializability is mainly of two types-

1. Conflict Serializability
2. View Serializability

[2.1] Conflict Serializability

- If a given schedule can be converted into a serial schedule by swapping its non-conflicting operations, then it is c as a **conflict serializable schedule**.
- Conflict serializability defines two instructions of two different transactions accessing the same da item to perform a read/write operation.
- It deals with detecting the instructions that are conflicting in any way and specifying the order in which the instructions should execute in case there is any conflict.
- A conflict serializability arises when one of the instruction is a write operation.

The following rules are important in Conflict Serializability,

1. If two transactions are both read operation, then they are not in conflict.

2. If one transaction wants to perform a read operation and other transaction wants to perform a write operation, then they are in conflict and cannot be swapped.

3. If both the transactions are for write operation, then they are in conflict, but can be allowed to take place any order, because the transactions do not read the value updated by each other.

How to check for Conflict Serializable Schedule ?

Conflict Equivalence and Conflict Serializable Schedule

Before we discuss conflict equivalence and conflict serializable schedule, you must know about conflicts. So, what is a conflict is described in the following section.

What is a conflict?

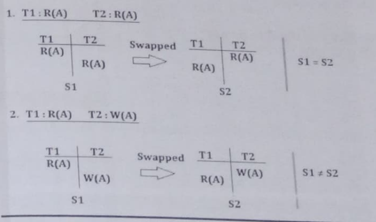
A pair of Operations in a schedule such that if their order is interchanged then the behavior of atle one of the transactions may change.

Operations are conflict, if they satisfy all three of the following conditio

- They belong to different transactions
- They access the same data item
- At least one of the operation is a write operation.

For example:

Swapping is possible only if S1 and S2 are logically equal.



1. T1 : R(A)   T2 : R(A) → Non Conflict Pair
2. T1 : R(A)   T2 : W(A) → Conflict Pair

1. R1(A);W2(A) :	It is a conflict pair, Because T1 is reading initial value of A and T2 writing a different value for A and if we interchange their order then T1 will read the value of A written by T2 , thus behavior will change.
2. W1(A); R2(A) 3. W1(A); W2(A)	Conflict Pairs
4. R1(A); R2(A)	Not a Conflict Pair
5. R1(A); W2(B) 6. W1(A); R2(B)	Not a Conflict Pairs because they are acting on two different values, A and B.
7. R1(A); W1(A)	We can't interchange order within same transaction. Therefore, No conflict

Conflict Serializable Schedule

A Schedule is conflict serializable if it is conflict equivalent to any of serial schedule.

Testing for conflict serializability

Method 1:

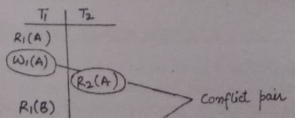
- First write the given schedule in a linear way.
- Find the conflict pairs (RW, WR, WW) on same variable by different transactions.
- Whenever conflict pairs are find, write the dependency relation like Ti → Tj, if conflict pair is from Ti to Tj. For example, (W1(A), R2(A)) ⇒ T1 → T2
- Check to see if there is a cycle formed,
  - If yes= not conflict serializable
  - No= we get a sequence and hence are conflict serializable.

Example:

Question: Check whether the schedule is conflict serializable or not?

S: R1(A); W1(A); R2(A); R1(B); W1(B); R2(B)

Solution:



Conflict Pairs	Dependency Relation

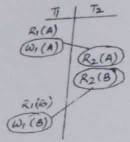
W1(A); R2(A) T1 → T2

W1(B); R2(B); T1 → T2

No Cycle formed. Therefore, Conflict Serializable Schedule.

Question: Check whether the schedule is conflict serializable or not?
S: R1(A); W1(A); R2(A); R2(B); R1(B); W1(B)

Solution :



CONFLICT PAIRS:-

W1(A); R2(A)
R2(B); W1(B)

DEPENDENCY RELATION:-

$T_1 \to T_2$
$T_2 \to T_1$

cycle dependency

Hence cycle formed ∴ Not a Conflict serializable schedule.

Practice Problems based on Conflict Serializability-

- Practice these problems based on **checking whether a given schedule is conflict serializable or not.**
- These problems are important from examination point of view.

Problem-01:

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

S : $R_1(A)$, $R_2(A)$, $R_1(B)$, $R_2(B)$, $R_3(B)$, $W_1(A)$, $W_2(B)$

Solution-

Step-01:

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

- $R_2(A)$, $W_1(A)$ ($T_2 \to T_1$)
- $R_1(B)$, $W_2(B)$ ($T_1 \to T_2$)
- $R_3(B)$, $W_2(B)$ ($T_3 \to T_2$)

Step-02:

$$\begin{tikzcd}
T1 \arrow[rr, bend left] & & T2 \arrow[ll] \\
& T3 \arrow[ur] &
\end{tikzcd}$$

Draw the precedence graph-

Since, there exists a cycle in the precedence graph, therefore, the given schedule S is not conflict serializable.

Problem-02:

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

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

Solution-

Checking whether schedule S is conflict serializable or not-

Step-01:

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

- $R_2(X)$, $W_3(X)$ ($T_2 \to T_3$)
- $R_2(X)$, $W_1(X)$ ($T_2 \to T_1$)
- $W_3(X)$, $W_1(X)$ ($T_3 \to T_1$)
- $W_3(X)$, $R_4(X)$ ($T_3 \to T_4$)
- $W_1(X)$, $R_4(X)$ ($T_1 \to T_4$)
- $W_2(Y)$, $R_4(Y)$ ($T_2 \to T_4$)

Step-02:

Draw the precedence graph-

$$\begin{tikzcd}
T1 \arrow[rr, bend left] & T2 \arrow[l] \arrow[r] \arrow[d] & T4 \\
& T3 \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Since, there exists no cycle in the precedence graph, therefore, the given schedule S is conflict serializable.

Checking whether schedule S is recoverable or not-

Because conflict serializable schedules are always recoverable, therefore above schedule S is definitely recoverableAlternatively, because there exists no dirty read operation as all the transactions which updates the values immediately commits, therefore S is recoverable.

Also, S is a cascadeless schedule.

Problem-03:

Check whether the given schedule S is conflict serializable or not. If yes, then determine all the possib serialized schedules-

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

Solution:- Checking whether schedule S is conflict serializable or not-

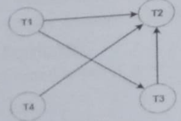
Step-01:

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

- $R_4(A)$, $W_2(A)$ $(T_4 \rightarrow T_2)$
- $R_3(A)$, $W_2(A)$ $(T_3 \rightarrow T_2)$
- $W_1(B)$, $R_3(B)$ $(T_1 \rightarrow T_3)$
- $W_1(B)$, $W_2(B)$ $(T_1 \rightarrow T_2)$
- $R_3(B)$, $W_2(B)$ $(T_3 \rightarrow T_2)$

Step-02:

Draw the precedence graph-



Since, there exists no cycle in the precedence graph, therefore, the given schedule S is conflict serializab

[2.2] View Serializability:-

- If a given schedule is found to be view equivalent to some serial schedule, then it is called as a vi **serializable schedule**.

- View serializability is the another type of serializability.
- It can be derived by creating another schedule out of an existing schedule and involves the same transactions.

Example : Let us assume two transactions T1 and T2 that are being serialized to create two different sch SH1 and SH2, where T1 and T2 want to access the same data item. Now there can be three scenarios

1. If in SH1, T1 reads the initial value of data item, then in SH2 , T1 should read the initial value of that data item.

2. If in SH2, T1 writes a value in the data item which is read by T2, then in SH2, T1 should write the va the data item before T2 reads it.

3. If in SH1, T1 performs the final write operation on that data item, then in SH2, T1 should perform the final write operation on that data item.

If a concurrent schedule is view equivalent to a serial schedule of same transaction then it is said to be **View serializable.**

PRACTICE PROBLEMS BASED ON CHECKING WHETHER THE GIVEN SCHEDULE IS VIEW SERIALIZABLE OR NOT-

Before you proceed to solve these questions, make sure that you have gone through the previous article which discusses the concept of **View Serializability and Steps to check whether the given schedule is view serializable or not.**

Problem-01:

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

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

Solution-

We know, if a schedule is conflict serializable, then it is surely view serializable. So, let us check whether the given schedule is conflict serializable or not.

Checking for conflict serializability-

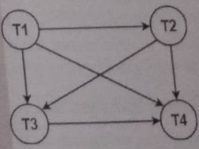
Step-01:

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

- $W_1(B)$, $W_2(B)$ $(T_1 \rightarrow T_2)$
- $W_1(B)$, $W_3(B)$ $(T_1 \rightarrow T_3)$
- $W_1(B)$, $W_4(B)$ $(T_1 \rightarrow T_4)$
- $W_2(B)$, $W_3(B)$ $(T_2 \rightarrow T_3)$
- $W_2(B)$, $W_4(B)$ $(T_2 \rightarrow T_4)$
- $W_3(B)$, $W_4(B)$ $(T_3 \rightarrow T_4)$

Step-02:

Draw the precedence graph-



Since, there exists no cycle in the precedence graph, therefore, the given schedule S is conflict serial

Therefore, **S is view serializable**.

Problem-02:

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

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

Solution-

We know, if a schedule is conflict serializable, then it is surely view serializable. So, let us check whethe given schedule is conflict serializable or not.

Checking for conflict serializability-

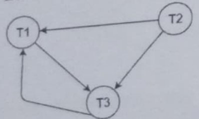
Step-01:

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

• $R_1(A)$, $W_3(A)$	$(T_1 \rightarrow T_3)$
• $R_2(A)$, $W_3(A)$	$(T_2 \rightarrow T_3)$
• $R_2(A)$, $W_1(A)$	$(T_2 \rightarrow T_1)$
• $W_3(A)$, $W_1(A)$	$(T_3 \rightarrow T_1)$

Step-02:

Draw the precedence graph-

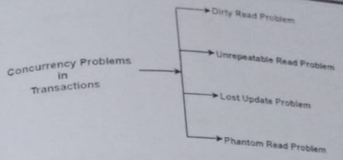


Since, there exists a cycle in the precedence graph, therefore, the given schedule S is not conflict serializable.Since, the given schedule S is not conflict serializable, therefore, it may or may not be view serializable.

[3] Concurrency Control:-

Concurrency Problems in Transaction-

When multiple transactions execute concurrently in an uncontrolled or unrestricted manner, then it might to several problems.Such problems are called as **concurrency problems**. These concurrency problems a


Concurrency Problems in Transactions
Dirty Read Problem
Unrepeatable Read Problem
Lost Update Problem
Phantom Read Problem

1. Dirty Read Problem
2. Unrepeatable Read Problem
3. Lost Update Problem
4. Phantom read Problem

1. Dirty Read Problem-

Reading the data that is written by an uncommitted transaction is known as **dirty read**.

OR

If a transaction reads some value from buffer that is written by an uncommitted transaction, then this rea known as **dirty read**.

The read is known as dirty read because there is always a chance that the uncommitted transaction might back in later stages and thus causing the other transaction to read a value that actually does not even exis

Note-

It is very important to note that the dirty read does not lead to inconsistency always. It becomes proble only when the uncommitted transaction fails and roll backs due to some reason.

Example-

Transaction T1	Transaction T2
R (A)	
W (A)	
	R (A) // Dirty Read
	W (A)
	Commit
Failure	

T1 reads a value A and updates it in buffer.

1. T2 reads the updated value from buffer and gets committed.
2. T1 fails in later stages and rolls back

Thus, T2 will have read the value that is now never considered to be existed which leads to inco the database.

If the transaction T1 would not have failed later i.e. if it would have successfully committed, th not have been any inconsistency created in the database.

2. Unrepeatable Read Problem-

If a transaction gets to read unrepeated i.e. different values in different reads of the same vari has not updated the value of that variable, then this problem is known as **Unrepeatable Read**

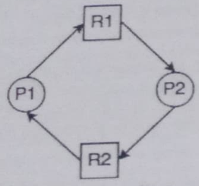


Fig. Deadlock State

In the above diagram,

- Process P1 holds Resource R2 and waits for resource R1, while Process P2 holds resource R1 and waits for Resource R2. So, the above process is in deadlock state.
- There is the only way to break a deadlock, is to abort one or more transactions. Once, a transaction is aborted and rolled back, all the locks held by that transaction are released and can continue their execution. So, the DBMS should automatically restart the aborted transactions.

Deadlock Conditions
Following are the deadlock conditions,
1. Mutual Exclusion
2. Hold and Wait
3. No Preemption
4. Circular Wait

A deadlock may occur, if all the above conditions hold true.

In Mutual exclusion states that at least one resource cannot be used by more than one process at a time. The resources cannot be shared between processes.

Hold and Wait states that a process is holding a resource, requesting for additional resources which are being held by other processes in the system.

No Preemption states that a resource cannot be forcibly taken from a process. Only a process can release a resource that is being held by it.

Circular Wait states that one process is waiting for a resource which is being held by second process and the second process is waiting for the third process and so on and the last process is waiting for the first process. It makes a circular chain of waiting.
Deadlock Prevention
Deadlock Prevention ensures that the system never enters a deadlock state.

Following are the requirements to free the deadlock:

1. No Mutual Exclusion : No Mutual Exclusion means removing all the resources that are sharable.

2. No Hold and Wait : Removing hold and wait condition can be done if a process acquires all the resources that are needed before starting out.

3. Allow Preemption : Allowing preemption is as good as removing mutual exclusion. The only need is to restore the state of the resource for the preempted process rather than letting it in at the same time as the preemptor.

4. Removing Circular Wait : The circular wait can be removed only if the resources are maintained in a hierarchy and process can hold the resources in increasing the order of precedence.
Deadlock Avoidance
- Deadlock Avoidance helps in avoiding the rolling back conflicting transactions.
- It is not good approach to abort a transaction when a deadlock occurs.
- Rather deadlock avoidance should be used to detect any deadlock situation in advance.