(1) Why Concurrency Control is needed ? Demonstrate with example (10 Marks) (OR) What are the anomalies occur due Interlear execution (OR) Explain the following i) The lost update problem ii) Temporary update problem (OR) The problems that occur when Concurrent execution is uncontrolled.

1) Lost update problem:

This problem occur when two transaction that access the same database items have their operation interleaved in a way that makes the value of some database item incorrect

for example - Consider following transaction,

1) Salary of employee is read during transaction T_1.
2) Salary of employee is read by another transaction T_2.
3) During transaction T_1, the salary is incremented by ₹ 200
4) During transaction T_2, the salary is incremented by ₹ 500

This update is lost

Time

only this update is successful

T_1	T_2	
Read		Salary = ₹1000
	Read	Salary = ₹.1000
update Increment salary by ₹ 200		Salary = ₹1200
	update Increment Salary by ₹ 500	Salary = ₹ 1500

This result of the above sequence is the update

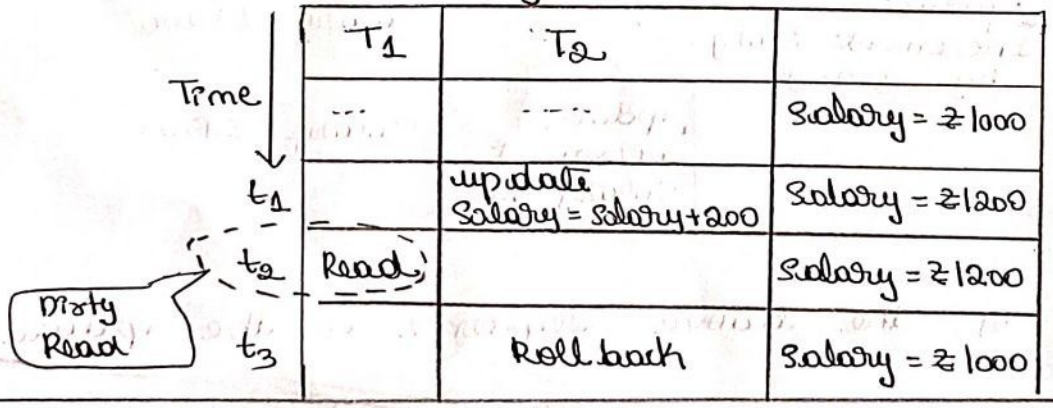
by transaction T_1 is completely lost. Therefore this problem is called as lost update problem.

2) Dirty read or uncommited read problem or temporary update

The dirty read is a situation in which one transaction reads the data immediately after the write operation of previous transaction

T_1	T_2
R(A)	
A = A+50	
W(A)	
	R(A)
	A = A-20
	W(A)
	Commit
Commit	

For example - Consider following transactions.
Assume initially salary is = ₹1000.



T_1	T_2	
...	...	Salary = ₹1000
	update Salary = Salary+200	Salary = ₹1200
Read		Salary = ₹1200
	Roll back	Salary = ₹1000

1) At the time t_1, the transaction T_2 updates the salary to ₹1200.
2) This salary is read at time t_2 by transaction T_1. obviously it is ₹1200
3) But at the time t_3, the transaction T_2 performs rollback by undoing the changes made by T_1 and T_2 at time t_1 and t_2
4) Thus the salary again becomes = ₹1000. This situation leads to Dirty Read or uncommited Read because here the read made at time t_2 (immediately after update of another transaction) becomes a dirty read

3) Non - repeatable read problem

This problem is also known as inconsistent analysis problem. This problem occurs when a particular transaction sees two different values for the same row within its life time.

For example -

	T_1	T_2	
Time t_1	Read		Salary = ₹1000
t_2		update salary from ₹1000 to ₹1200	Salary = ₹1000
t_3	- -	Commit	
t_4	Read		Salary = ₹1200

3) Then at time t_3, the transaction T_2 get committed

4) Now when the transaction T_1 reads the same salary at time t_4, it gets different value than what it had read at time t_1, Now, transaction T_1 cannot repeat its reading operation. Thus inconsistent value are obtained.

(4) Incorrect read problem :

This is a problem in which one of the transaction makes the changes in the database system and due to these changes another transaction can not read the data item which it has read just recently

For example.

	T_1	T_2	
t_1	Read		Salary = ₹1000
t_2		Read	Salary = ₹1200
t_3	Delete Salary		No Salary
t_4		Read	Salary = ₹1200

Time ↓

(1) At time t_1, the transaction T_1 reads the value of salary as ₹1000

(2) At time t_2, the transaction T_2 reads the value of the same salary as ₹1000

(3) At time t_3, the transaction T_1 deletes the variable Salary.

(4) Now at time t_4, when T_2 again reads the salary it gets error. Now transaction T_2 can not identify the reason why it is not getting the salary value which is read just few time back

(2) Explain the ACID properties of a database transaction and system log. (06 marks)

or

What is transaction? discuss the desirable properties of Transaction (05 marks)

In a database, each transaction should maintain ACID property to meet the consistency and integrity of the database.

1) Atomicity :-

* This property states that each transaction must be considered as a single unit and must be completed fully or not completed at all.
* No transaction in the database is left half completed.
* Database should be in a state either before the transaction execution or after the transaction execution. It should not be in a state 'executing'.
* For example :- In above mentioned withdrawal of money transaction all the five steps must be completed fully or none of the step is completed. Suppose if transaction gets failed after step 3, then the customer will get the money but the balance will not be updated accordingly. The state of database should be either at before ATM withdrawal (i.e, customer without withdrawn money) or after ATM withdrawal. This will make the system in consistent state.

2) Consistency :-

* The database must remain in consistent state after performing any transaction.
* For example :- In ATM withdrawal operation, the balance must be updated appropriately after performing transaction. Thus the database can be in consistent state.

3) Isolation :-

* In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system.
* No transaction will affect the existence of any other transaction.
* For example :- If a bank manager is checking the account balance of particular customer, then manager should see the balance either before withdrawing the money or after withdrawing the money. This will make sure that each individual transaction is completed and any other dependent transaction will get the consistent data out of it. Any failure to any transaction will not affect other transaction in this case. Hence it makes all the transactions consistent

4) Durability :-

* The database should be strong enough to handle any system failure.

* If there is any set of insert / update, then it should be able to handle and commit to the database.
* If there is any failure, the database should be able to recover it to the consistent state.
* For example:- In ATM withdrawal example, if the system failure happens after customer getting the money then the system should be strong enough to update database with his new balance after system recovers. For that purpose the system has to keep the log of each transaction and its failure. So when the system recovers, it should be able to know when a system has failed and if there is any pending transaction, then it should be updated to database.

Properties of System log :-

* Log or Journal keeps track of all transaction operations that affect the values of database items
* one main memory buffers hold the last part of the log file, so that log entries are first added to the main memory buffer.
* when the log buffer is filled, or when certain other conditions occur, the log buffer is appended to the end of the log file on disk.
* In additional, the log is periodically backed up to archival storage (tape) to guard against such catastrophic failures.

* The following are the types of entries - called log records - that are written to the log file and the corresponding action for each log record.
* In these entries, T refers to a unique transaction -id that is used to identify each transaction.

1. [start_transaction, T]. Indicates that transaction T has started execution.
2. [write_item, T, X, old_value, new_value]. Indicates that transaction T has changed the value of database item X from old_value to new_value.
3. [read_item, T, X]. Indicates that transaction T has read the value of database item X.
4. [commit, T]. Indicates that transaction T has completed successfully, and affirms that its effect can be committed to the database.
5. [abort, T]. Indicates that transaction T has been aborted.

Explain why a transaction execution should be atomic. Explain ACID properties by considering the following

T_1 : read(A); A := A - 50;

write(A);

Read(B);

B = B + 50;

write(B);

→ By considering the below example,

read(A)
A := A - 50
write(A)
read(B)
B := B + 50
write(B)

Atomicity:

Either all operations of the transaction are properly reflected in the database or none are

According to this defination,

In the above example consider the A = 100 and B = 200 initially, Now read(A) i.e read A value 100 from database, Now update this value i.e A = A - 50.

A = 100 - 50

A = 50.

Now updated A value is 50.

2) Consistancy:

Before Update	After update
A = 100.	A = 150. (A = 100-50)
B = 200.	B = 250. (B = 200+50)
A+B = 100+200	A+B = 50+250
A+B = 300	A+B = 300

Here before and after also A+B = 300 remain same. ∴ It proves that both the values are in consistant.

3) Isolation:

When multiple transaction may execute concurrently, each transaction must be unaware of other concurrently executing transactions. This is intermediate transaction results must be hidden from other concurrently executed trasactions.

T_1	T_2	
read (A)		
A := A-50		
write (A);		
	read (A)	here
	read (B)	
	print (A+B) →	250 ≠ 300.
read (B)		
B := B+50		
write (B).		

Now read the B value from database
i.e B = 200.
Now update the B value as

$$B = B + 50$$
$$B = 200 + 50$$
$$\boxed{B = 250}$$

Now updated B value is 250.
So now A & B both are updated as

$$A = 50 \quad \& \quad B = 250.$$

So now it is necessary update or write these two values in the database. We should not update any one of the values into database. We should update either both the values or none. this defines the automicity.

ACID properties

$\Rightarrow$

```
T1 : read (A);
A := A - 50;
write (A);
read (B);
B := B + 50;
write (B).
```

1) Atomicity:

Before update
A = 100
B = 200

After update
A = A - 50 $\Rightarrow$ 100 - 50
A = 50
B = B + 50 $\Rightarrow$ 200 + 50
B = 250.

So now update both A & B values as 50 & 250.

4) Durability:

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

Before update in database	After update in local buffer	After commit command then Now in database
A = 100 B = 200	A = 50 B = 250.	A = 50 B = 250.

(4) With neat diagram, explain the various state of transaction execution.

or

Draw a state diagram and discuss the typical state that transaction goes through during execution.

A transaction is an atomic unit of work that should either be completed in its entirety or not done at all. For recovery purposes, the system keeps track of start of a transaction, termination, commit or aborts.

- BEGIN_TRANSACTION : marks the beginning of transaction execution
- READ or WRITE : specify read or write operations on the database items that are executed as part of a transaction.
- END_TRANSACTION: specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution
- COMMIT_TRANSACTION : signals a successful end of the transaction so that any changes executed by the transaction can be safely committed to the database and will not be undone.
- ROLLBACK : signals that the transaction has ended unsuccessfully, so that any changes or effects that the transaction may have applied to the data base must be undone.

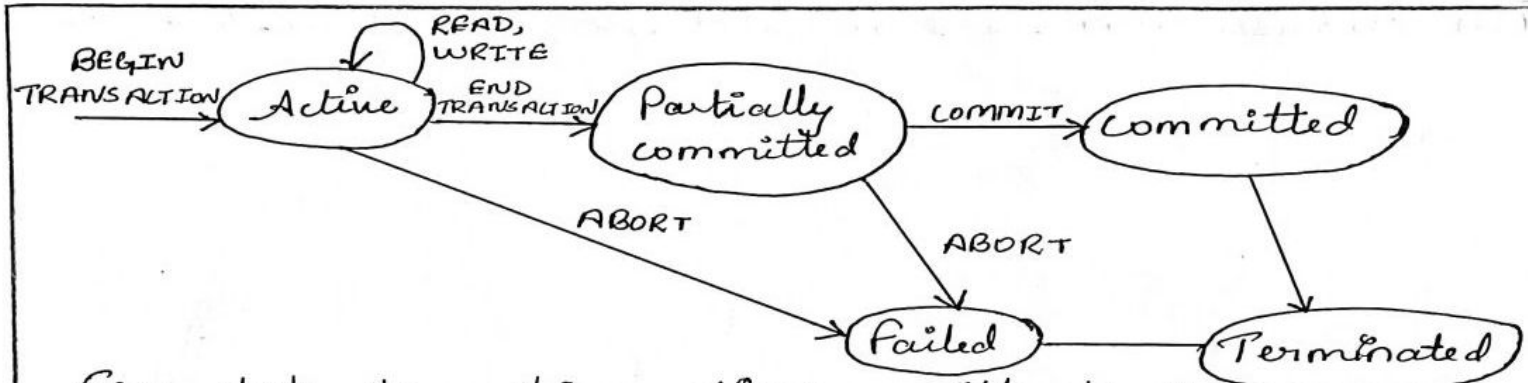


Fig: state transition diagram illustrating the states for transaction execution.

* A transaction goes into active state immediately after it starts execution and can execute read and write operations.
* When the transaction ends it moves to partially committed state.
* At this end additional checks are done to see if the transaction can be committed or not If these checks are successful the transaction is said to have reached commit point and enters committed state. All the changes are recorded permanently in the db.
* A transaction can go to the failed state if one of the checks fails or if the transaction is abort -ed during its active state. The transaction may then have to be rolled back to undo the effect of its write operation.
* Terminated state corresponds to the transaction leaving the system. All the information about the transaction is removed from System table.

(5) How to check Conflict Serializibility of Schedule. Explain with an Example 10M

Step 1:- TO check whether the schedule is conflict Serializable or not we will check from top to bottom. Thus we will start reading from top to bottom as.

$T_1 : R(A) \rightarrow T_1 : W(A) \rightarrow T_2 : R(A) \rightarrow T_2 : R(B) \rightarrow T_1 : R(B) \rightarrow T_1 : W(B)$

[Refer table at last]

Step 2:- We will find Conflicting operations. Two operation are called as conflicting operations if all the following conditions hold true for them-

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

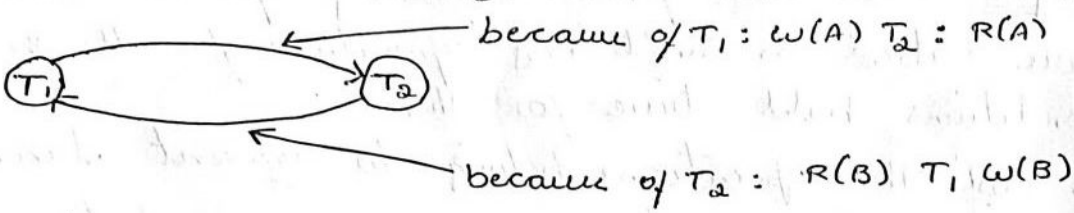
from above given example in the top to bottom scanning we find the conflict as $T_1 : W(A) \rightarrow T_2 : R(A)$

1. Here note that there are two different transactions T_1 and T_2
2. Both work on same data item i.e A and
3. One of the operation is write operation.

Step 3:- We will build a precedence graph by drawing one node from each transaction. In above given scenario as there are two transactions. there will be two nodes namely T_1 and T_2.

Step 4 :- Draw the edge between conflicting transactions, for example in above given scenario, the conflict occurs while moving from T_1 : W(A) to T_2 : R(A). Hence edge must be from T_1 and T_2.

Step 5 :- Repeat the step 4 while reading from top to bottom, finally the precedence graph will be as follows



Step 6 : check if any cycle exist in the graph. Cycle is a path using which we can start from one node and reach to the same node. If the cycle is found then Schedule is not Conflict Serializable. In the steps we get a graph with cycle. that means give schedule is not Conflict Serializable.

Table :

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(B)
R(B)	
W(B)	

(6) What is Schedule ? Explain Conflict Serailizibility of Schedule with Example.

Schedule is an order of multiple transactions executing in concurrent environment.

Conflict Serializability :

In the defination three conditions are specified for a conflict in conflict serializability -

* there should be different transactions.
* the operations must be performed on same data items.
* One of the operation must be the write (W) operation

→ we can test a given schedule for conflict serializability by constructing a precedence graph for the schedule and by searching for absence of cycles in the graph.

→ predence graph is a directed graph. consisting of $G_1 = (V, E)$ where V is set of vertices and E is set of edges. The set of edges consist of all edges $T_i - T_j$ for which one of three conditions hold :

1. T_i Executes write (Q) before T_j Executes read (Q)
2. T_i Executes read (Q) before T_j Executes write (Q)
3. T_i Execute write (Q) before T_j executes write (Q).

- A Serializability order of the transactions can be obtained by finding a linear order consistent with partial order of the precedence graph. this process is called topological sorting.

Testing for Serializability.

following method is used for testing the Serializability: To test the conflict Serializibity we can draw a graph $G = (V, E)$ where V = Vertices which represent the number of transactions.

E = Edge for conflicting pairs:

Step 1: Create a node for Each transaction

Step 2: find the conflicting pair (RW, WR, WW) on the same variable (or data item) by different transactions

Step 3:- Draw edge for the given schedule. Consider following cases

1. T_i Execute write (Q) before T_j Execute read (Q). then draw edge from T_i to T_j
2. T_i execute read (Q) before T_j Execute write (Q). then draw edge from T_i to T_j
3. T_i executes write (Q) before T_j Execute write (Q). then draw edge from T_i to T_j

Step 4:- now, if precedence graph is cyclic then it is a non conflict serializable schedule and if the precedence graph is a acyclic then it is conflict Serializable schedule.

Checking for Serializability:-

Refer the previous question (5)

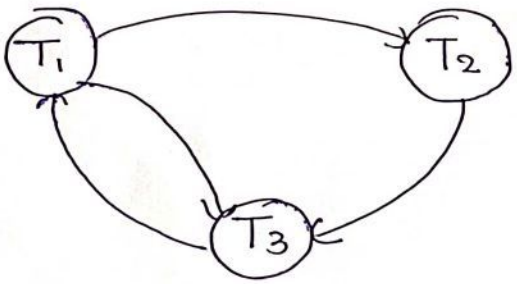
7) Illustrate with precedence graph, which of the following schedules conflict serializable (06 Marks)

a) R1(X) ; R3(X) : W1(X) ; R2(X) ; W3(X);

b) R3(X) ; R2(X) ; W3(X) ; R1(X); W1(X);

a) R1(X) : R3(X) : W1(X) ; R2(X) ; W3(X)

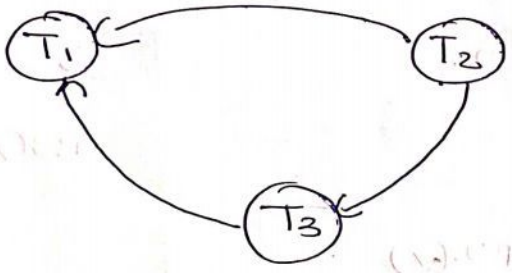
T1	T2	T3
R1(X)		
		R3(X)
W1(X)		
	R2(X)	
		W3(X)



It is not serializable

(b) $r_3(x)$; $r_2(x)$; $w_3(x)$; $r_1(x)$; $w_1(x)$

T1	T2	T3
		$r_3(x)$
	$r_2(x)$	
		$w_3(x)$
R1(x)		
w1(x)		



It is a serializable

8) Check wheather the below Schedule is conflict Serializable or not.

{B2, r2(X), b1, r1(X), w1(X), r1(Y), w1(Y), w2(X), e1, c1, e2, c2}

b2 and b1 represents beging transaction 2 and begin transaction 1. Similarly, e1 and e2. represents end transaction 1 and end transaction 2.

We will rewrite the Schedule as follows

T_1	T_2
	r2(X)
r1(X)	
w1(X)	
r1(Y)	
w1(Y)	
	w2(X)

Step 1 :- We will find conflicting operations. Two operations are called as conflicting operations if all the following conditions hold true for them.

i) Both the operations belong to different transactions

ii) Both the operations are on some data item

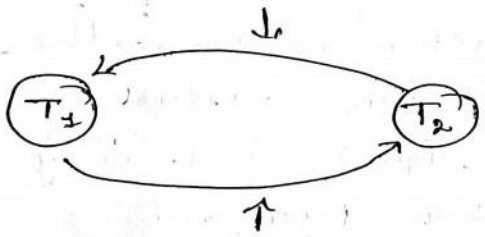
iii) At least one of the two operations is a write operation

The conflicting entries are as follows :-

T_1	T_2
	$R_2(X)$
$R_1(X)$	
$W_1(X)$	
$R_1(Y)$	
$W_1(Y)$	
	$W_2(X)$

Step 2 :- Now we build a precedence graph for conflicting entries.

$T_2 : R_2(X)$ $T_1 : W_1(X)$



$T_1 : W_1(X)$, T_2 $W_2(X)$

As there are two transactions only two nodes are present in the graph

Step 3 :- We get a graph with cycle, that means given schedule is not conflict Serializable.

9) What is the two-phase locking protocal? How does it Guarantee Serializability? 6 marks.

* The two phase locking is a protocol in which there are two phases :-

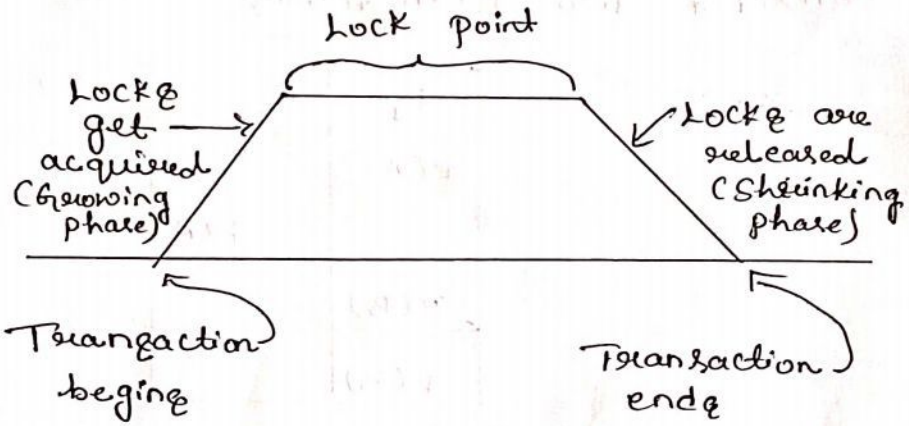
i) Growing phase (Locking phase) :- It is a phase in which the transaction may obtain locks but does not release any lock.

ii) Shrinking phase (unlocking phase) :- It is a phase in which the transaction may release locks but does not obtain any new lock.

* Lock point :- The last position of first unlock position is called lock point.

E.x :-

LOCK (A)
LOCK (B)
LOCK (C)
- - -
- - -
- - -
Lock point
unlock (A)
Unlock (B)
Unlock (C).



The important rule for being a two phase locking is - all lock operation preced unlock operation. If t1 is in two-phase locking mode but t2 is not in two phase this means t2 as shared lock which can be access by any data item red & it is locked again this

we get lock - , unlock - lock sequence, not feasible in 2-pha

* Serialization is mainly an issue of handling write operation. Because any inconsistency may only be created by write operation.
* Multiple reads on a database item can happen parallely
* 2-phase locking protocal restricts this unwanted read / write by applying exclusive lock.
* Moreover, when there is an exclusive lock on an item it will only be released in shrinking phase. Due to this restriction there is no chance of getting any inconsistent state.

The Serializability using two phase locking can be understood with the help of following example.

Consider two transactions

T_1	T_2
R(A)	
	R(A)
R(B)	
R(B)	

Step 1 :- Now we will apply two phase locking. That means we will apply locks in growing and shrinking phase

T_1	T_2
Lock-S(A)	
R(A)	
	Lock-S(A)
	R(A)
Lock-X(B)	
R(B)	
W(B)	
unlock-X(B)	
	unlock-S(A)

Note that above schedule is Serializable as it prevents interference b/w two transactions.

The Serializability order can be obtained based on the lock point. The lock point is either last lock operation position or first unlock position in the transaction.

The last lock position is in T_1, then it is in T_2. Hence the Serializability will be $T_1 > T_2$ based on lock points. Hence the Seriability Sequence can be $R_1(A)$; $R_2(A)$; $R_1(B)$; $W_1(B)$

Similar questions - with an algorithm explain two phase locking

Defines two phase locking technique for concurrency Control

10) what is 2PL? Explain with an example.

or

Define locking Protocol. Describe the Strict Two-phase Locking (2PL) Protocol.

Locking Protocol:- It is a mechanism in which a transaction cannot read or write data unless the appropriate lock is acquired.

Strict two phase locking :-

The Strict 2PL Protcol is a basic two phase Protocol but all the exclusive mode be held until the transaction commits. That means in other words all the exclusive locks & unlocked only after the transaction is committed. That also means that if T_1 has exclusive lock, then T_1 will release the exclusive lock only after Commit operation, then only other transactions is allowed to read or write. For ex:- Consider two transactions

T_1	T_2
W(A)	
	R(A)

If we apply the locks then

T_1	T_2
Lock-X(A)	
W(A)	
Commit	
unlock(A)	
	Lock-S(A)
	R(A)
	unlock-'S(A)

Thus only after commit operation in T_1, we can unlock the exclusive lock. This ensures the Strict Serializability

Thus compared to basic two phase locking protocol. the advantage of Strict 2PL Protocol is it ensures Strict Serializability.

11) Explain ARIES Recovery algorithm with Example (8 marks)

:- Algorithms for Recovery and Isolation Exploiting Semantics

ARIES is a Recovery algorithm designed to work with a no force, steal database approach.

It is based on three concepts:

1. write-ahead logging
2. repeating history during redo, and
3. logging changes during undo

1. write-ahead logging : Any change to an object is first recorded in the log and the log must be written to stable storage before changes to the object are written to disk.

2. Repeating history during Redo : On restart after a crash, ARIES retraces the actions of a database before the crash and brings the system back to the exact state that it was in before the crash. Then it undoes the transactions still active at crash time.

3. Logging changes during undo :- Changes made to the database while undoing transactions are logged to ensure such an action isn't repeated in the Event of repeated restarts.

- The ARIES recovery procedure consists of three main steps:
  1. Analysis phase
  2. REDO phase

3& UNDO phase

- The analysis phase :- identifies the dirty (updated) pages in the buffer and the set of transaction at the time of crash
- the appropriate point in the log where the REDO operation should start is also determined

The REDO phase :-

* Reapplies updates from the log to the database
* Certain information in the ARIES log will provide the start point for REDO, from which REDO operations are applied until the end of the log is reached

The UNDO phase :-

* The log is scanned backward and the operations of transactions that were active at the time of the crash are undone in reverse order.
- the information needed for ARIES to accomplish its recovery procedure includes the log, the transaction table, and the Dirty page table. Additionally, checkpointing is used.
- these tables are maintained by the transaction manager and written to the log during checkpointing.
* In ARIES, every log record has an associated log sequence number (LSN) that is monotonically increasing and indicates the address of the log record on disk
* Each LSN corresponds to a specific change (action) of some transaction

- Beside the log, two table are needed for efficient recovery: The transaction table and Dirty page Table. which are maintained by the transaction manager.
- when a crash occure. there table are rebuilt in the analysis phase of recovery.

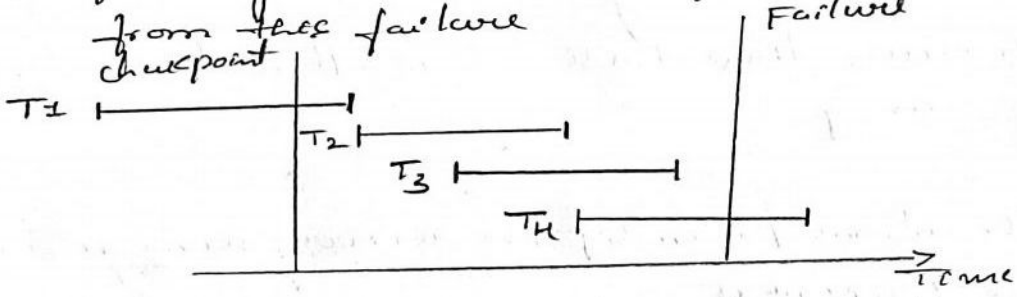
---

12) Describe the actions taken by the recovery manager during checkpoint (6 marks)

- The checkpoint is a type of mechanism where all the previous logs are removed from system and permanently stored on the storage disk.
- The checkpoint is like a bookmark. While the execution of the transaction, such checkpoints are marked, and the transaction is executed then using the steps of the transaction, the log file will be created.
- When it reaches to the checkpoint, then the transaction will be updated into the database, and till that point, the Entire log file will be removed from the file. Then the log file is updated with the new step of transaction till next checkpoint and so on
- The checkpoint is used to declare one point before which the DBMS was in the consistent state, and all transactions were committed.

Recovery using checkpoint

In the following manner, a Recovery System recovers the database from the failure



- The Recovery System reads log file from the End to Start. it reads log file from T_H to T_1
- Recovery System maintains two Lists, a redo-list, and an undo-List
- The transaction is put into redo state of the recovery System sees a log with <Tn, Start> and <Tn, Commit> or just <Tn, Commit>. In the redo-List and their previous List, all the transactions are removed and then Redone before saving their logs.

for Example: In the log file. transaction T2 and T3 will have <Tn, Start> and <Tn, Commit>. The T_1 transaction will have only <Tn, Commit> in the log file. That's why the transaction is committed after the checkpoint is crossed. Hence puts T1, T2 and T3 transaction into redo List

- The transaction is put into undo state if the recovery System sees a log with <Tn, Start> but no commit or abort log found. In the undo-List, all the transactions are undone, and their logs are removed

13) Discuss the immediate-update recovery technique in both single-user and multiple user environments. Explain the Database recovery Techniques.

In this technique, the database is updated during the execution of transaction even before it reaches to its commit points.

If the transaction gets failed before it reaches to its commit point, then the Rollback operation needs to be done to bring the database to its earlier consistent state. That mean the effects of operations need on the database. for that purpose both Redo and undo operations are both required during the recovery. This technique is known as UNDO|REDO technique.

e.x:- Consider two transaction T_1 and T_2 as follows

T_1	T_2
Read(A,a)	Read (C.c)
a = a-10	C = c-20
write (A.a)	write (C.c)
Read (B.b)	
b = b+10	
write (B.b)	

Here T_1 and T_2 are executed Serially. Initially A = 100, B = 200 and C = 300.

If the crash occurs

ii) Just after write (B.b) iii) Just before write (C.c) iii) Just after $\langle T_2, \text{commit} \rangle$

Then using the immediate database modification approach the result of above three scenarios can be elaborated as follows:

The contents of log and database is a follows:

Log	Database
$\langle T_1, \text{Start} \rangle$	
$\langle T_1, A, 100, 90 \rangle$	$A = 90$
$\langle T_1, B, 200, 210 \rangle$	$B = 210$
$\langle T_1, \text{commit} \rangle$	
$\langle T_2, \text{Start} \rangle$	
$\langle T_2, C, 300, 280 \rangle$	
$\langle T_2, \text{commit} \rangle$	$C = 280.$

The recovery scheme uses two recovery techniques.

i) UNDO (T_i) : The transaction T_i needs to be undone if the log contains $\langle T_i, \text{Start} \rangle$ but does not contain $\langle T_i, \text{commit} \rangle$. In this phase, it restores the values of all data items.

ii) REDO (T_i) : The transaction T_1 needs to be redone if the log contains both $\langle T_i$ Start$\rangle$ and $\langle T_1$, Commit$\rangle$. In this phase, the data item values are set to the new values as per the transaction. After a failure has occurred log record is consulted to determine which transaction need to be redone.

a) Just after write (B.b) :- When system comes back from this crash. it sees that there is $\langle T_1$. Start$\rangle$ but not $\langle T_i$, commit$\rangle$. Hence T_1 must be be undone. That means old values of A & B are restored. Thus old values of A & B are taken from log and both the transaction T_1 and T_2 are re-executed.

b) Just after write C :- Here both the redo and undo operations will occur.

c) UNdo :- when System comes back from this crash, it sees that there is $\langle T_2$, Start$\rangle$ but no $\langle T_2$, Commit$\rangle$. Here T_2 must be undo. That means old values of C is restored.

d) Redo :- The transaction T_1 must be done as log contains both the $\langle T_1$, Start$\rangle$ and $\langle T_1$, Commit.

So A = 90, B = 210 and C = 300

e> Just after <T_2, commit> :- when a System comes back this from this crash, it sees that there are two transaction T_1 and T_2 will both Start and commit points. That means T_1 and T_2 need to redone. So A = 90, B = 210 and C = 280.

14) Discuss the UNDO and REDO Operations and the recovery techniques that use each (6M)

Deferred Database Modification

* In this technique, the database is not updated immediately
* only log file is updated on each transaction
* when the transaction reaches to its commit point, then only the database is physically updated from the log file.
* In this technique, if a transaction fails before reaching to its commit point, it will not have changed database anyway. Hence there is no need for the UNDO operation. the REDO operation is required to record the operation from log file to physical database. Hence deferred database modification techniques is also called as NO UNDO/REDO algorithm

i. UNDO(T_i): the transaction T_i needs to be undone if the log contains $<T_i, start>$ but does not contain $<T_i, commit>$. In this phase, it restores the values of all data items updated by T_i to the old values

ii REDO(T_i) :- the transaction T_i needs to be redone if the log contains both $<T_i, start>$ and $<T_i, commit>$ In this phase, the data item values are set to the new values as per the transaction. After a failure has occurred log record is consulted to determine which

transaction need to be redone.

for example :-

Consider two transactions T_1 and T_2 as follows:

T_1	T_2
Read (A, a)	Read (C, c)
a = a - 10	C = C - 20
write (A, a)	write (C, c)
Read (B, b)	
b = b + 10	
write (B, b)	

If T_1 and T_2 are executed serially with initial value of A = 100, B = 200 and C = 300, then the state of log and database if crash occurs

a) Just after write (B, b)

b) Just after write (C, c)

c) Just after $< T_2$, commit >

the result of above three scenarios is as follows:

Initially the log and database will be

log	Database
$< T_1$, Start >	
$< T_1$, A, 90 >	
$< T_1$, B, 210 >	
$< T_1$, Commit >	A = 90
$< T_2$, Start >	B = 210
$< T_2$, C, 280 >	
$< T_2$, Commit >	C = 280

a. Just after write (B.b)

* Just after write operation. no commit record appears in log. Hence no write operation is performed on database. So database retains only old values. Hence A = 100 and B = 200 respectively.
* Thus the System Come back to original position and no redo operation take place.
* the incomplete transaction of T_1 can be deleted from from log.

b. Just after write (C. x)

* The state of log records is as follows:
* Note that crash occurs before T_2 commits. At this point T_1 is completed Successfully. So new values of A and B are written from log to database. But as T_2 is not Committed. there is no redo (T_2) and the incompleted transaction T_2 can be deleted from log.
* The redo (T_1) is done as $< T_1$. Commit > gets Executed. Therefore A = 90. B = 210 and C = 300 are the values for database.

c. Just after $< T_2$. Commit >

The log records are as follows:

$< T_1$. Start >

$< T_1$. A. 90 >

$< T_1$. B. 210 >

$\langle T_1, \text{Commit} \rangle$

$\langle T_2, \text{Start} \rangle$

$\langle T_2, 6, 280 \rangle$

$\langle T_2, \text{Commit} \rangle$

$\longleftarrow$ Crash occurs here

clearly both T_1 and T_2 reached at commit point and then crash occurs. So both undo(T_1) and redo(T_2) are done and update values will be A=90, B=210, C=280

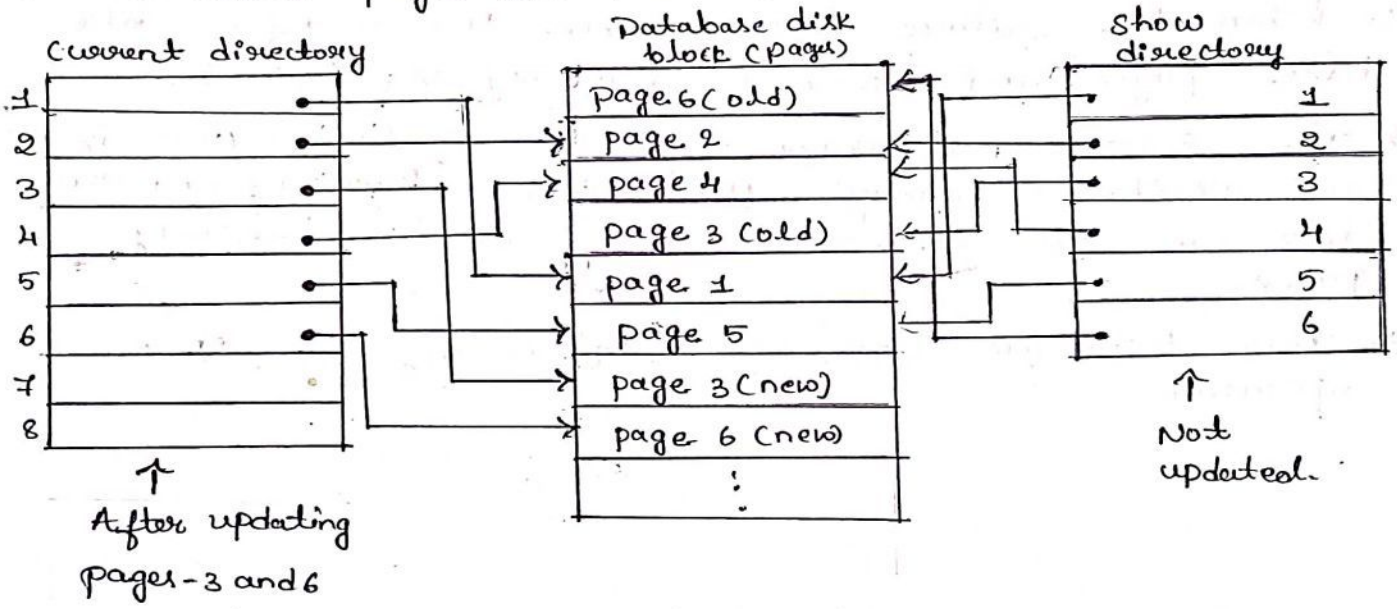
15) Explain how Shadow Paging helps to cover from transaction failure.

or

Explain the various database recovery techniques with examples. 5 or 8 marks.

* Shadow Paging is a recovery schema in which database is considered to be made up of number of fixed size disk pages.

* A directory or a Page table is constructed with n number of Pages where each i^{th} page points to the i^{th} database page on the disk.



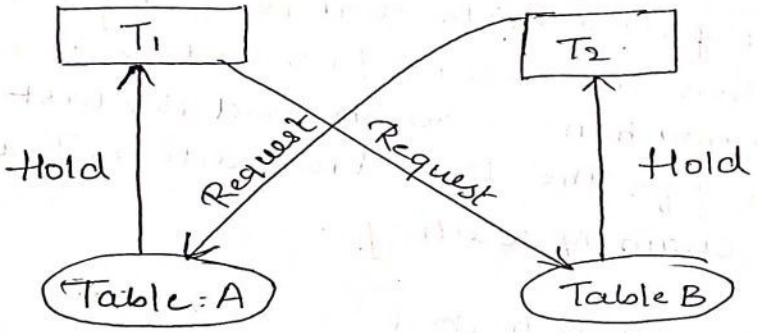
* The directory can be kept in the main memory
* when a transaction begins executing, the current directory - whose entries point to the most recent or current database on disk - is copied into a another directory called shadow directory.
* The shadow directory is then saved on disk while the current directory is used by the transaction.

* During the execution of transaction, the shadow directory is never modified.
* when a write operation is to be performed then the new copy of modified database page is created but the old copy of database page is never over-written. This newly created database page is written somewhere else.
* The current directory will point to newly modified web page and the shadow page directory will point to the old web page entries of database disk.
* when the failure occurs then the modified database pages and current directory is discarded.
* The state of database before the failure occurs is now available through the shadow directory and this state can be recovered using shadow directory pages.
* This technique does not require any UNDO/REDO operation.

16) When deadlock and starvation problem occur? Explain how these problem can be resolved.

⟹ "A system is in deadlock state if there exists a set of transactions' such that every transaction in the set is waiting for another transaction in the set."

For example: Consider that transaction T_1 holds a lock on some rows of table A and needs to update some rows in the B table. Simultaneously transaction T_2 holds locks on some rows in the B table and needs to update the rows in the A table held by transaction.



- Now transaction T_1 is waiting for T_2 to release its lock and similarly, transaction T_2 is waiting for T_1 to release its lock. All activities come to a halt state and remain at a standstill. This situation is called deadlock in DBMS.

Conditions for a Deadlock to occur

There are four conditions for a deadlock to occur. A deadlock may occur if all the following conditions holds true.

1. Mutual exclusion condition: There must be at least one table that cannot be used by more than one transaction at a time.

2. Hold and wait condition: A transaction that is holding a table can request for additional table that are being held by other transactions in the system.

3. No preemption condition: A table cannot be forcibly taken from a transaction. Only the transaction can release a table that is being held by it.

4. Circular wait condition: A condition where one transaction is waiting for a table that is being held by second transaction, and second transaction is waiting for third transaction --- so on and the last transaction is waiting for the first transaction. Thus making a circular chain of waiting.

Deadlock Prevention method

There are two techniques for deadlock prevention.

① Wait-Die: Suppose there are two transactions T_1 and T_2 and let TS(T) is a timestamp of any transaction T. If T_2 holds a lock by some other transaction and T_1 is requesting for resources held by T_2 then the following actions are performed by DBMS:

* Check if $TS(T_i) < TS(T_j)$ - if T_i is the older transaction and T_j has held some resource, then T_i is allowed

to wait until the data-item is available for execution. That means if the older transaction is waiting for a resource which is locked by the younger transaction, then the older transaction is allowed to wait for resource until it is available.

* Check if $TS(T_i) < TS(T_j)$: If T_i is older transaction and has held some resource and if T_j is waiting for it, Then T_j is killed and restarted later with the random delay but with the same timestamp.

(2) Wound-wait : If wound wait scheme, if the older transaction requests for a resource which is held by the younger transaction, then older transaction forces younger one to kill the transaction and release the resource. After some delay, the younger transaction is restarted but with the same timestamp.

* If the older transaction has held a resource which is requested by the younger transaction, then the younger transaction is asked to wait until older releases it.

Deadlock Detection method

Deadlock detection is done using wait for graph method.

Wait for graph:

In this method, a graph is created based on the transaction and their lock. If the created graph has a cycle or closed loop, then there is a deadlock.

* The wait for the graph is maintained by the system for every transaction which is waiting for some data held by the others. The system keeps checking the graph if there is any cycle in the graph.
* This graph consists of a pair $G = (V, E)$, where V is a set of vertices and E is a set of edges.
* The set of vertices consists of all the transactions in the system.

When transaction T_i requests a data item currently being held by transaction T_j then the edge T_i to T_j is inserted in the wait for graph. This edge is removed only when transaction T_j is no longer holding a data item needed by Transaction T_i.

Example: Consider the following transactions, we will draw a wait for graph for this sinario and check for deadlock.

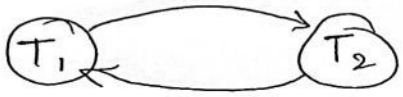
T_1	T_2
R(A)	
	R(A)
W(A)	
R(B)	
	W(A)
W(B)	

Rule 1: If T_1 has Read operation and then T_2 has Write operation then draw an edge from T_1 to T_2

Rule 2: If T_1 has Write operation and then T_2 has Read Operation then draw an edge from T_1 to T_2

Rule 3 : If T_1 has write operation and then T_2 has write operation then draw an edge from T_1 to T_2

T_1	T_2
R(A)	
	R(A)
W(A)	
R(B)	
	W(A)
W(B)	



As cyclic is detected in the wait for graph there is no need to further process. The deadlock is present in this transaction scenario.

17) B) What is Time stamp? How does system generates Time stamp?

* The time stamp ordering protocol is a schema in which the order of transaction is decided in advance based their timestamps.
* Thus it schedules are serialized according to their time stamps.
* The timestamp-ordering protocol ensures that any conflicting read and write operations are executed in timestamp order
* A larger transaction indicates a more recent transaction or it is also called as 'younger transaction while lesser timestamp indicates older transaction
* Assume a collection of data items that are accessed, with read and write operations by transaction
* For each data item X the DBMS maintain the following values;
  - RTS(X): The timestamp on which object X was last read (by some transaction T_i, i.e, RTS(X) := TS(T_i))

    [NOTE: RTS stands for Read Time stamp]

• WTS(X): The timestamp on which object x was last written (by some transaction T_i, i.e, WTS(X) := TS(T_i))
[Note: WTS stands for write Time Stamp]

Basic Timestamp ordering Algorithm:

Case 1 (Read): Transaction T issues a read(X) operation

i) If TS(T) < WTS(X), then read(X) is rejected. T has to abort and be rejected

ii) If WTS(X) <= TS(X), then execute read(X) of T and update RTS(X).

Ex:- Suppose we have two transaction T_1 and T_2 with timestamp 10 sec. and 20 sec respectively

10 sec T_1	20 sec T_2
R(X)	
	W(X)
R(X)	

RTS(X) and WTS(X) is initially = 0
Then RTS(X) = 10, when transaction T_1 executes
After that WTS(X) = 20 when transaction T_2 executes

Now if read operation R(x) occurs on transaction T_1 at $TS(T_1) = 10$ then $TS(T_1)$ i.e. $10 < WTS(x)$ i.e 20, hence we have to reject second read operation on T_1 i.e

10 sec T_1	20 sec T_2
R(x)	
	W(x)
(W(x))	

This write operation gets rejected as it occurs at older timestamp than the write operation of transaction 2

Case 2 (write): Transaction T issues a write ~~read~~(x) operation

i) If $TS(x) < RTS(x)$ or if $TS(T) < WTS(x)$, then write is rejected.

ii) If $RTS(x) <= TS(T)$ or $WTS(x) <= TS(T)$, then execute write(x) of T and update WTS(x).

Ex: Suppose we have two transactions T_1 and T_2 with Time stamps 10 sec. and 20 sec respectively.

10 sec T_1	20 sec T_2
R(x)	
	W(x)
W(x)	

RTS(x) and WTS(x) is initially = 0

Then RTS(x) = 10 when transaction T_1 executes after that WTS(x) = 20 when transaction T_2 executes

Now if write operation W(x) occurs on transaction T_1 at TS(T_1) = 10 then

TS(T_1) i.e 10 < WTS(x) hence we have to reject second ~~ope~~ write operation on T_1 i.e

10 sec T_1	20 sec T_2
R(x)	
	W(x)
(W(x))	

This write operation gets rejected as it occurs at older timestamp than the write operation at transaction 2

ii) Suppose we have two transaction T_1 and T_2 with timestamp 10 sec and 20 sec respectively

10 sec T_1	20 sec T_2
W(x)	
	W(x)

RTS(x) and WTS(x) is initially =0
then WTS(x) = 10 as transaction T_1 executes
Now if write operation W(x) occurs on transaction T_2 at TS(T_2) = 20 then
TS(T_2) i.e 20 > WTS(x) which is 10, hence we accept write operation on T_2, The transaction T_2 will perform write operation and new WTS will updated as
WTS(x) = 20

Case 1 (read):

ii) Suppose we have two transaction T_1 and T_2 with time stamps 10 sec and 20 sec respectively

10 sec T_1	20 sec T_2
W(x)	
	R(x)

RTS(x) and WTS(x) is initially =0.
then WTS(x) = 10 as transaction T_1 executes
now if ~~write~~ read operation R(x) occurs on transaction T_2 at TS(T_2) = 20 then

TS(Ta) i.e 20 > WTS(x) which is 10, hence we accept read operation on Ta the transaction Ta will perform read operation and now RTS will be updated as

RTS(x) = 20

18) Describe transaction support in SQL. Give example (08 Marks)

(OR)

Describe the four level in isolation in SQL (06 Marks)

→ ① using SQL statements various transaction occur with database system.

② A single SQL statement is always atomic.

③ with SQL there is no explicit Begin Transaction statement

④ Every transaction must have explicit End statement It can be wether COMMIT (OR) ROLLBACK

⑤ Every transaction must have certain characteristics which are specified using SET Transaction.

⑥ Three characteristics

1. Access Mode
2. Diagnostic Area size.
3. Isolation Level.

① Access Mode

* In SQL, the access mode can be specified as READ ONLY or READ WRITE
* The default is READ WRITE unless the isolation level of READ UNCOMITED is specified, in which case READ ONLY is assumed

② Diagnostic Area size

* the diagnostic area size n, specifies an integer value n, indicating the number of conditions that can be held simultaneously in the diagnostic area

③ Isolation Level

* The transaction should take place in such a way that it is the only transaction that is accessing the resources in a database system at particular instance

* Isolation levels defines the degree to which a transaction must be isolated from the data modifications made by any other transaction in the database system

* The isolation level is denoted as <Isolation>.

(i) Serializable :-

- This is the highest isolation level.
- Serializable execution is defined to be an execution of operation in which concurrently executing transactions appears to be serially executing
- This is a default isolation level.

ii) Repeatable Read :-

- This is the most restrictive isolation level.
- The transaction holds read locks on all rows it references.
- It holds write locks on all rows it inserts, updates, or references
- Since other transaction cannot read, update or delete these rows, it avoids non repeatable read

iii) Read Committed :

- This isolation level allows only committed data to be read.
- Thus it does not allows dirty read (i.e one transaction reading of data immediately after written by another transaction).
- The transaction hold a read or write lock on the current row, and thus prevent other rows from reading, updating or deleting it.

iv) Read uncommitted

- It is lowest isolation level.
- In this level, one transaction may read not yet committed changes made by other transaction.
- This level allows dirty reads.
- In this level transactions are not isolated from each other.

1) Dirty Read: The dirty read is a situation in which one transaction reads the data immediately after the write operation of pervious transaction.

2) Non Repeatable Read: Allowing another transaction to write a new value after multiple reads of one transaction is called non-repeatable read.

3) Phantoms: This is a problem in which one of the transaction makes the change in the database system & due to these changes another transaction can not read the data item which it has read just recently.

Isolation level	Types of Problem		
	Dirty Read	Non-repeatable Read	Phantoms
Read uncommitted	Yes	Yes	Yes
Read Commited	No	Yes	Yes
Repeatable Read	No	No	Yes
Serializable	No	No	No