19/08

Module - IV

Transaction

* Group a set of tasks into a single execution unit.
* A database transaction, by definition, must be atomic, consistent, isolated and durable. These are popularly known as ACID properties
* Transaction has only 2 results: success or failure
* Atomic : anyone of action should take place (one should share & other should recieve)
* Consistent : After transaction, the total should remain the same, before as before transaction.

A	B	
1000	1000	(atomic)
900	1100	2000
		200

* Isolated : Data isolated only for us
  No interruption feel.
* Durability : ~~No change~~ Once it is committed, ~~after~~ if any system failure occurs, the commited data should not change.

Commit → permanently saved.

Ex: Transfer RS.100 from A to B

```
BEGIN TRANSACTION
R(A);      //Read the balance of A
A=A-100;
W(A);      //write the updated value, R(B)
R(B);
B=B+100;
W(B);
END TRANSACTION
```

Begin & End
↳ Transaction Boundaries.

Terminology

* BEGIN TRASACTION
  - Beginning of transaction execution.

· END TRASACTION
  • Transaction execution completed.

· ROLLBACK_TRANSACTION (ABORT)
  · This signals that the transaction has ended unsuccessfully so that any changes made by transaction that are undone.

· COMMIT_TRANSACTION
  · signals successful end of the transactions.
  · All changes made by transaction are recorded permanently.

ACID Properties

1. Atomicity: The entire transaction takes place at once or doesn't happen at all.

2. Consistency: The database must be consistent before and after the transaction.

3. Isolation: - Muetiple transactions occur independently without interference.

4. Durability: The changes of a successful transaction occurs even if the system failure occurs.

Single-users Vs Multicuser Systems

* Single - User System:
  • At most one user at a time can use the system.

* Multiuser System:
  • Many users can access the system concurrently.

* Concurrency:
  • Interleaved processing: - Concurrent execution of processes is interleaved in a single CPU. (One after other)
    (interleaved.

- Parallel processing :- Processes are concurrently executed in multiple CPUs. ~~(multiple CPU ones)~~

Simple Model of a Database Transaction.

- A database is a collection of named data items.
- Granularity of data
  - The size of data items.
  - concepts are independent of data granularity.

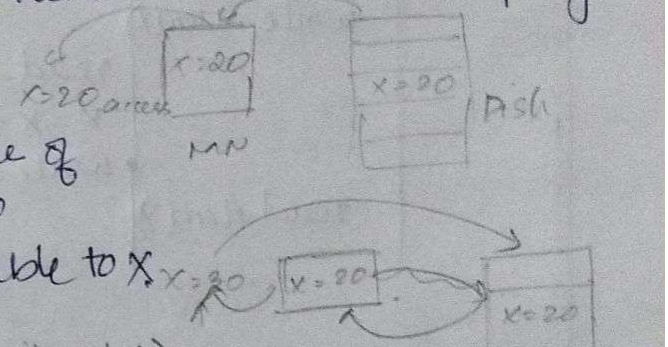
Granules :- diff parts.

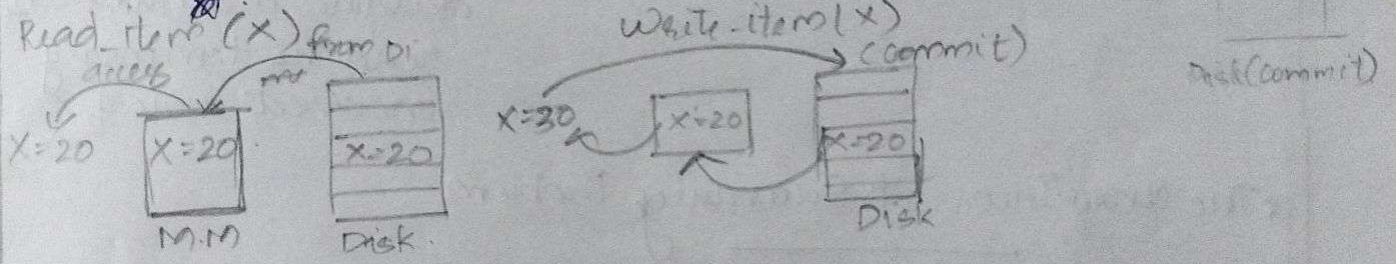
Granularity :- diff size of parts

- Basic DB operations Read & write.

  read_item(X): Reads a DB item named X into a program variable also named X.

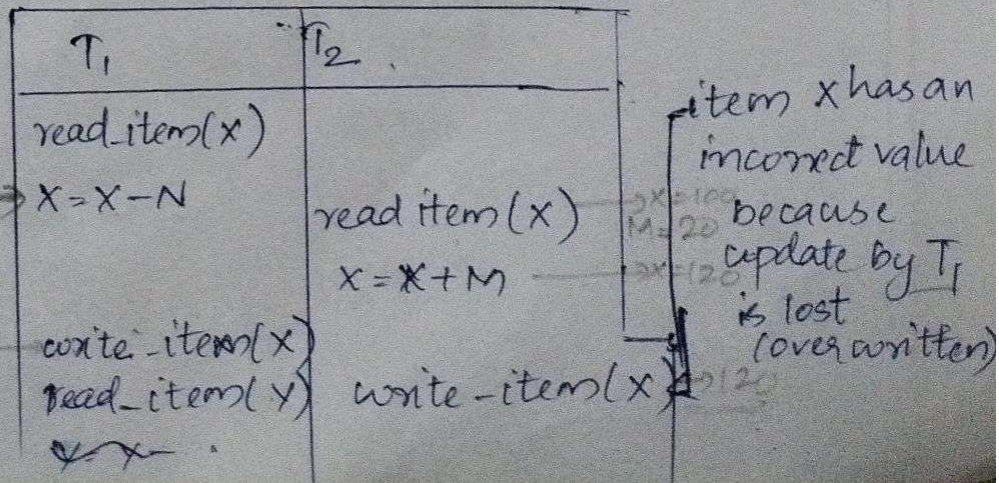
  write_item(X): writes the value of the program variable X to database naming the variable to X.





Why Concurrency Control is Needed

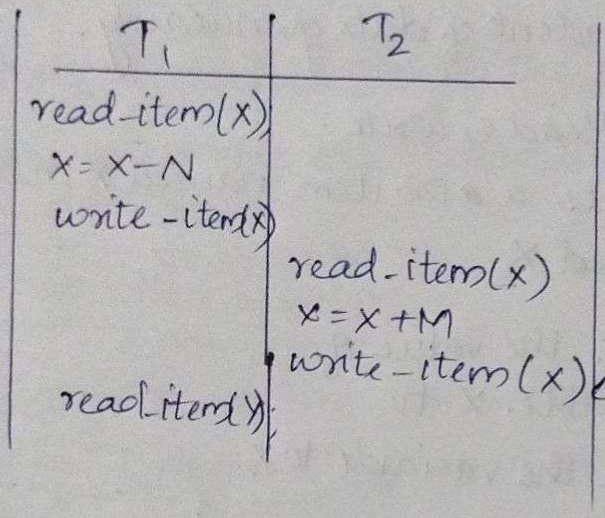
* The Lost Update Problem.
  - This occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect.



*The Temporary Update (or Dirty Read) Problem

*This occurs when one transaction updates a database item and then the transaction fails for some reason.

*The updated item is accessed by another transaction before it is changed back to its original value.



Transaction T_1 fails and must change the value of X back to its old value: meanwhile T_2 has read the temporary incorrect value of X.

*The Incorrect Summary Problem

*If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated.

T_1 T_2

Sum = 0
Read_item(X)
X + N
write(X)

Sum = 0
Read_item(X)
Sum = X - N
write(X)

* The Unrepeatable Read Problem

* Similar to Summary Problem.
where
* A transaction T reads the same item twice and the item is changed by another transaction T between the two reads.
* Hence, T receives different values for its two reads of the same item.

why recovery is needed.

* what causes a transaction to fail
1. A computer failure (system crash)

2. A transaction or system error:
   - Logical error.

3. Local errors or exception conditions detected by the transactions.
   certain conditions necessitate cancellation of the transaction.

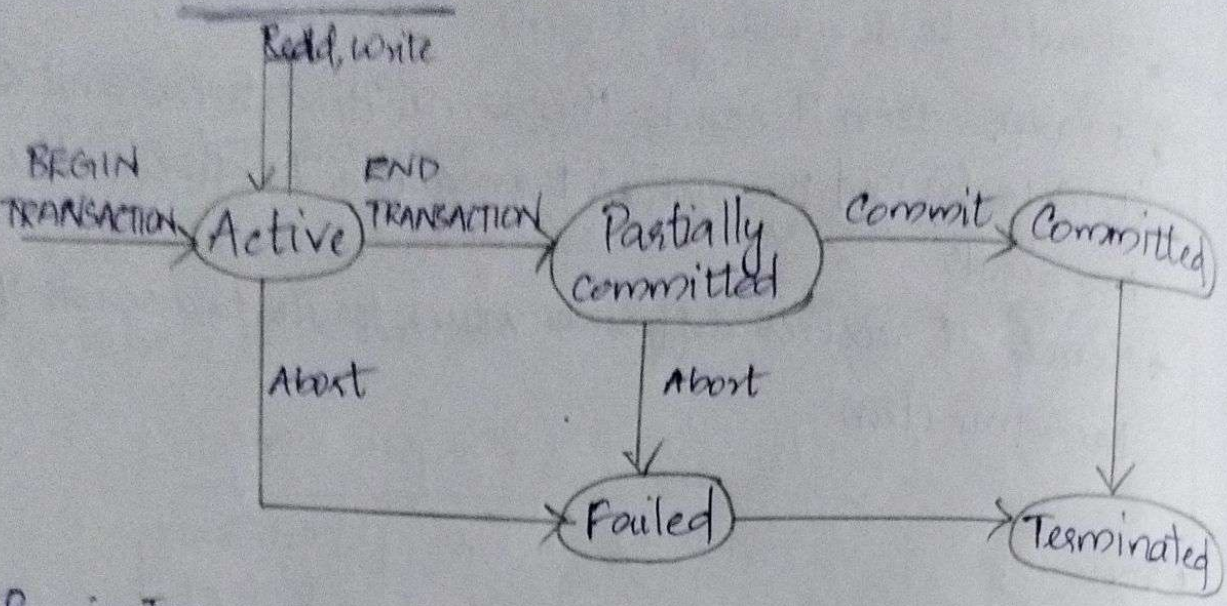
4. ~~Concur~~ Concurrency control enforcement

5. Disk failure: Lose of data may occur.

6. Physical problems and catastrophes

30/8

States of a Transaction



~~Begin Tra~~

BEGIN TRANSACTION: Transaction start occurs

ACTIVE: Current status of Transaction.
Read, write can be performed.

Partially Committed :- After end of transaction, the transaction is saved.

Committed :- Permanently stored after successfull transaction.

Terminated: End.

Failed :- Caused due to error in transaction.

Module - IV

Schedules (Histories) of Transactions

* Means history (what, when, how transaction occured)
* A schedule S of n transactions $T_1, T_2 \ldots T_n$ is an ordering of the operation of the transaction.
* When transactions are executing concurrently in an interleaved fashion, the order of execution of operations from the various transactions forms is known as schedules

3 types:-

1) Serial Schedule
2) Non-Serial Schedule
3) Serializable Schedule (comb of Serial & Non-serial)

Serial Schedule

T_1T_2 → order of performance

* Entire transactions are performed in serial order.
* Only one transaction is active at a time.
* Serial schedule : T_1T_2 → order of performance.

T_1		T_2
R(A)		
A = A+50	//150	
R(B)		
B = B-30	//170	
W(B)	//170	
W(A)	//150	
		R(A) //150
		A = A+40 //190
		W(A)
		R(B) //170
		B = B-60 //110
		W(B) //110

Initially:
A = 100
B = 200 } 300

Consistent.
Final:
A = 190
B = 110 } 300

Problems with serial schedules

- Poor resource utilization
- More waiting time (one after other)
- Less throughput
- Late response

Non Serial Schedule

- Executing the transaction in an interleaved or instructions of transactions are interleaved.

Schedule S1

T1	T2
R(A)	
A = A+50	
W(A) //150	
	R(A)
	A = A+40
	W(A) //190
R(B)	
B = B-30	
W(B) //170	
	R(B) //170
	B = B-60
	W(B) //110

Initially
A = 100
B = 200

Finally
A = ~~150~~ 190
B = 110

Gives same value of Serial schedule.

Schedule S2

T1	T2
R(A)	
A = A+50 //150	
	R(A) //100
	A = A+40
	W(A) //140
W(A) //150	
R(B) //200	
B = B-30 //170	
	R(B) //200
	B = B-60
	W(B) //140
W(B) //170	

Initially
A = 100
B = 200

Final
A = 150
B = 170

Not consistent

Serializable Schedule

* A non serial schedule of n transactions is said to be serializable if its equivalent to some serial schedule of same n transactions.
* Every serializable schedules are considered as correct
* Two schedules are equivalent, if:
  1) Result equivalence.
  2) conflict equivalence
  3) View equivalence

Result equivalence

* If they produce same final state of the database they produce same result

Ex: X = 100

S_1	S_2
read (x)	read (x)
x = x+10;	x = x*1.1
write(x) //110	write (x) //110

Possible if X = 100, but may vary if value of x varies, so result may not be equal

Conflict equivalence

- Conflict operations

Two operations in a schedule are said to conflict if they satisfy

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

Conflict: $R_1(x)$ $w_2(x)$ — No conflict: $R_1(x)$ $R_2(x)$

(R_1 → Read of T_1, w_1 → write of T_1)

$R_1(x)$	$w_2(x)$		
$R_2(x)$	$w_1(x)$	$w_2(x)$	$w_1(y)$
$w_1(x)$	$w_2(x)$	$R_1(x)$	$w_1(x)$

Conflict

* Two schedules are said to be conflict equivalent if the relative order of any two conflicting operations is the same in both schedules.

S_1		S_2	
T_1	T_2	T_1	T_2
R(A)		R(A)	
W(A)	→ R(A)	W(A)	→ R(A)
	W(A)	R(B)	
R(B)			W(A)
W(B)		W(B)	

* (min 2 conflict equi order must be there (for many))

(same order conflict)

S1 & S2 are conflict equivalent

R(A) W(A) → R(A) W(A) R(A) R(A) R(A) W(A) → R(A) W(A) R(A)

Not equivalent (conflict order diff)

* If conflict is not present in S1 & S2, then they are conflict equivalent.

Conflict Serializable Schedules

* If a non serial schedule is conflict equivalent to some serial schedule then it is called conflict serializable schedules

Schedule S_1		Serial Schedule S: T_1T_2	
T_1	T_2	T_1	T_2
R(A)		R(A)	
W(A)	→ R(A)	W(A)	
	W(A)	R(B)	
R(B)		W(B)	→ R(A)
W(B)	→ R(B)		W(A)
	W(B)		→ R(B)
			W(B)

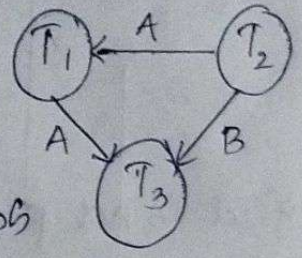
02/09

* To check a schedule is conflict ~~equivalence~~ serializable or not, we can use precedence graph.
* If the precedence graph is free from cycles (one cycle) (acyclic), then schedule is conflict serializable schedule.

Eg. Check if S_3 is conflict equivalent and ~~find~~ find its serializing order

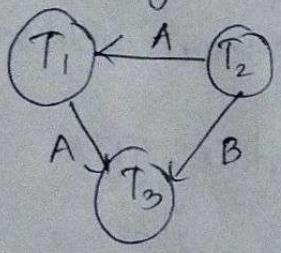
S_3

T_1	T_2	T_3
R(A)		
	W(B)	
		R(B)
	R(A)	
W(A)		
		R(A)



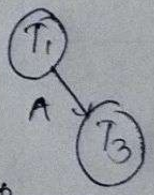
No cycle in precedence graph so S_3 is conflict serializable (serial schedule of the same order can be formed)

Serializing order : Order of Transaction.



Step1: Remove node with least indegree

T_2 → removed (First)



Step2: Remove Next node with least indegree

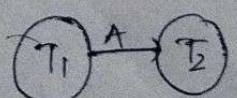
T_1 removed

T_3

Serializing Order: T_2 T_1 T_3.

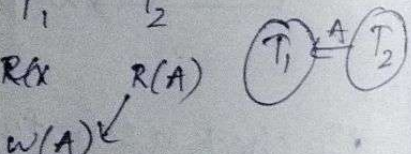
Rules for constructing Precedence Graphs

T_1	T_2
R(A)	
	W(A)

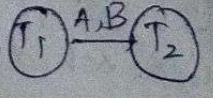


*only transaction and direction is mentioned

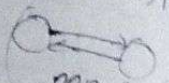
T_1	T_2
R(A)	
	W(A)



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



→ Conflict occurs from where only is specified (what type of conflict is not mentioned)



From the same order serial schedule can't be formed

Sd.

View Equivalence

*Two schedules S_1 and S_2 are said to be view equivalent if for each data item:-

* If the initial read in S_1 is done by T_i, then same transaction should be done in S_2 also.
* If the final write in S_1 is done by T_i, then in S_2 also T_i should perform the final write
* If T_i reads the item written by T_j in S_1, then S_2 also T_i should read the item written by T_j.

*A non serial schedule is view serializable schedule if it is view equivalent to some of its serial schedule.

*Every conflict serializable schedule is view serializable but is not true.

* Every conflict serializable schedule is serializable but vice versa not is not true

*Every view ~~serializab~~ serializable schedule is serializable and vice versa is also true.

Shortcut to check View Serializable or not

Step 1: All conflict serializable schedule are view serializable.

Step 2: For those schedules that are not conflict serializable

If there does not exist any blind ~~exists~~ writes, then the schedule is surely not view serializable.

//Blind write: performs a write operation without reading.

(If blind write, there is a chance of view serializable)

Not conflict serializable & Not blind write = Not view Equivalence.

Characterizing Schedule Based on Recoverability

Recoverability of Schedules

1. Recoverable and Irrecoverable schedule.
2. Cascadeless and Cascade schedule.
3. Strict and Non-strict schedule.

If transaction is committed it can't be rollback & abort.

Recoverable and Irrecoverable schedule

~~*If the transaction is committed after~~

*If a transaction first reads and writes a data, then it is committed first, then its a recoverable schedule.

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

Here, this is a recoverable schedule.

*If the transaction isn't committed after the first reading and writing, its called as irrecoverable schedule.

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
~~WCR~~	
commit;	

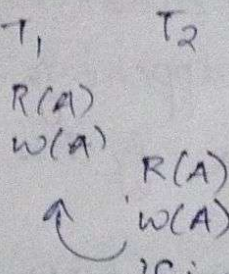
Irrecoverable.

Cascadeless and Cascade Schedule

Cascade Schedule:-

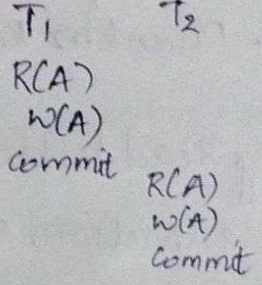
*Transactions are allowed to read uncommitted data from other transactions.

* this may lead to cascading rollback, which occur when one transaction fails and it forces other dependent transactions that have read its uncommitted data to also be rolled back.



Cascadeless Schedule

* Avoids cascading rollbacks by ensuring that a transaction only reads committed data.
* Means that no transaction can read data from another transaction until that data has been committed; so eliminates need of cascading rollbacks.



Strict and Non-Strict schedule

Strict schedule :-

* Provides guarantee of consistency and ensures recoverability by preventing transactions from working on uncommitted data.

*

Non-Strict

6/9 CONCURRENCY PROBLEMS IN TRANSACTIONS

Concurrency problem:
* occurs when multiple operations on transactions are happening at same time and are trying to access or modify the data same data

* Main types of concurrency problems:
1. Dirty Read problem
2. Unrepeatable Read problem
3. Lost update Problem
4. Phantom Read Problem (Incorrect Summary Problem)

Dirty Read Problem:
* Transaction reads the data that has been updated by another transaction that is still uncommitted.
* Arises due to multiple uncommitted transactions executing simultaneously.

T_1	T_2	
R(A)		
W(A)		
.	R(A)	// Dirty Read
.	W(A)	
Failure	commit	

Unrepeatable Read Problem:-
* Transaction reads the same data twice, but the data changes in between because another transaction updated it.
* Causes data to be inconsistent between the two reads.

T_1	T_2
R(x)	
	R(x)
w(x)	
	R(x)

* Phantom Read problem:

* Occurs when a transaction reads a variable once but when it tries to read that same variable again, an error occurs saying that the variable does not exist.

T_1	T_2
R(x)	
	R(x)
Delete(x)	
	R(x)

Lost Update Problem:-

* An update done to a data item by a transaction is lost as it is overwritten by the update done by another transaction.

T1	T2
R(x)	
X=X+15	
	R(x)
	X=X-25
	w(x)
w(x)	

S

id	Employee
1	Mark
3	Mary
100	Tony

T_1

Select * from Employee
where id between 1 and 3

T_2

Insert into Employee
values (2, "John")

Select * from employee
where id between 1 and 3

//Extra tuple is called a phantom tuple

23/09

Concurrency Control Protocols

* Serial schedule are correct but they are unacceptable.
* Solutions to concurrency problems :- (3 protocols)
  1. Lock based Protocols
  2. Time stamp based Protocols
  3. Graph based protocols

Lock Based Protocols

* Locking protocol means, whenever transaction wants to perform any operation (Read/write) on any data item, first it should request for lock on that data item.
* Once it acquire the lock then only it can perform the operation.
* Locking mechanism is managed by Concurrency Control Subsystem.

Transaction T_i wants to perform an operation on data item A

↓ Apply lock on data item (A).

Perform operations on A.

Shared Lock (S(A))

If a transaction wants to perform read operation on a data item, it should apply shared lock on that item.

T_1
S(A)
R(A)

Exclusive Lock (X(A))

If a transaction acquire this lock, it can perform read/write /both operation on that data item.

T_1
X(A)
R(A)
W(A)

Lock compatibility Matrix

* Concurrency control manager manages this.

T_i \ T_j	S(A)	X(A)
S(A)	YES	No
X(A)	No	No

Shared lock can be accessed but Exclusive lock can't be used.

Case I:-

* Multiple transactions can perform read operations on same data item.
* There can be multiple shared lock.

Case II:
If any transaction want to apply an exclusive lock on data item on which already a shared lock is allowed for some other transaction is Not possible.

T_1	T_2	T_3
S(A)		
R(A)	S(A)	X(A) // Not allowed
	R(A)	R(A)

Case I:
S(A) can be applied if unlocked.

T_1	T_2	T_3
S(A)		
R(A)		
U(A)	S(A)	X(A) // allowed.
	R(A)	W(A)
	U(A)	

Compatible or not?

T_1	T_2
S(A)	
R(A)	
S(B)	
R(B)	
U(A)	
	X(A)
	R(A)
	X(B) } → B not unlocked
	R(B)

Simple Locking protocol: If it follows all rules of protocol.

Problems with Simple Locking Protocol

* Sometimes does not guarantee Serializability
* may lead to deadlock.

To overcome

Two phase Locking (2PL)

* If every individual follows 2PL, then every schedule will be conflict serializable schedule.

A schedule is said to be in 2PL, if all transactions perform locking and unlocking in 2 phases:

1. Growing Phase (Locking only possible) (Strict)
   * New locks on data items can be again acquired but none can be unlocked.
   * Only locking is allowed no unlocking.
2. Shrinking Phase. (Optional phase) (If unlocked then it can't be reused)
   * Existing locks may be released but no new locks can be acquired
   * Only unlocking is allowed & no locking.

```
T1
Lock Apply (Growing Phase)
// perform operations.
Lock Release (Shrinking Phase).
```

To check whether a schedule is 2PL: check whether a data item is recused after unlocked. If ~~used~~ recused then its not in 2PL.

Lock point: The point at which a transaction acquires last lock.

T_1	T_2
X(A)	
R(A)	
W(A)	
S(B) //LP	
R(B)	
	S(B) //LP
	R(B)
	U(B)
U(A)	
U(B).	

Schedule is in 2PL
Lock compatible
(~~Un~~After unlocking ~~lock~~ not requested)
Check T_1 & T_2 wise

Problems with Basic 2PL.

1. Does not ensure Recoverability.
2. Cascading rollbacks
3. Deadlock are possible.

T_1	T_2
S(B)	
X(A) //LP	
R(B)	
	S(B) //LP
	R(B)
R(A)	
W(A)	
U(A)	
	X(A)
	R(A)
	U(A)
	C;
C;	

* 2PL
* Lock compatible :-
  *
* Irrecoverable

Deadlocks are possible.

T_1	T_2
X(A)	X(B)
R(A)	R(B)
W(A)	W(B)
*****	*****
wants to update B	wants to update A
*****	*****

* Unless B is released T_1 can't update B

Variation of 2PL

1. Strict 2PL
2. Rigorous 2PL
3. conservative 2PL.

Strict 2PL

* Follow basic 2PL + All exclusive locks should unlock only after commit operation.

Benefits:- * Generates conflict serilizable.
* Strict, recoverable, cascadeless schedule generated.

Drawback: * Deadlocks are possible.

T_1	T_2
X(A)	
R(A)	
S(B)	
R(B)	S(B)
	R(B)
W(A)	
U(B)	
C;	
U(A) ——→	X(A)
	R(A)
	W(A)
	C;

Lock compatible, 2PL, Strict 2PL.

Exclusive lock is committed before unlocking.

Rigorous 2PL

* Stronger than strict 2PL.
* Follow basic 2PL + All locks (both exclusive & shared locks) should unlock only after commit operation.
* Every rigorous 2PL is strict 2PL but vice versa is not true.

~~Benifits~~:

Read Drawbacks: Deadlocks

T_1	T_2
S(A)	
R(A)	
	S(B)
	R(B)
	X(C)
	R(C)
	W(A)
	C; //Commit
	U(B)
	U(C)
S(C)	
R(C)	
C; //Commit	

Rigorous 2PL & Strict 2PL

Conservative 2PL (C2PL)

* No deadlock.
* Follow basic 2PL + All transaction should obtain all lock they need before the transaction begins.
* Release all lock after commit.
* Recoverable, serializable, cascadeless, Deadlock free
* Not practical to implement.