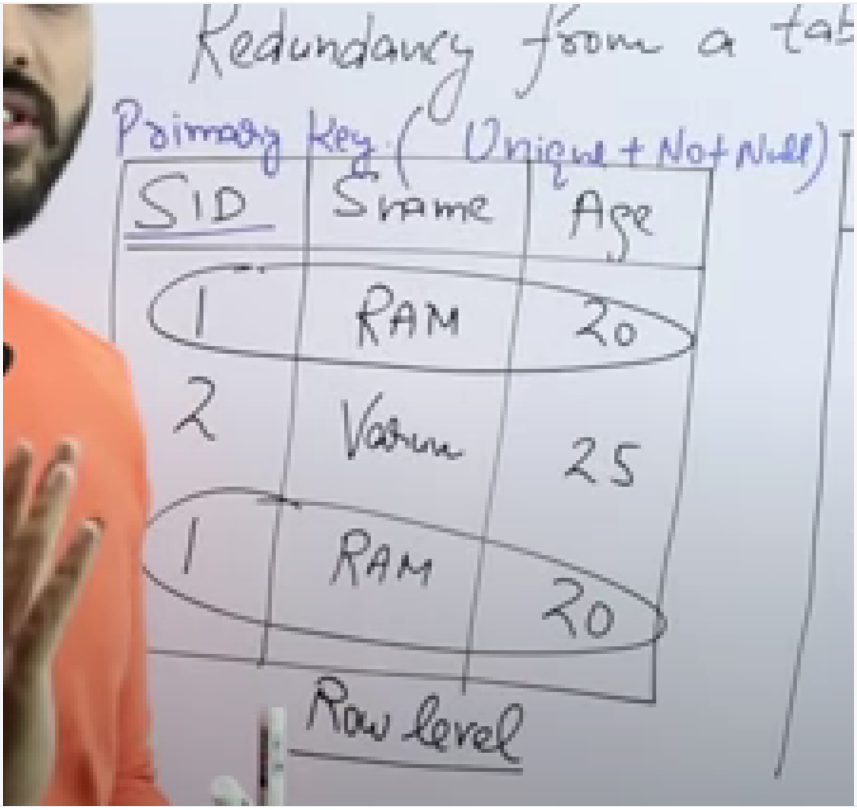
Normalization

it is a technique to remove or reduce the redundancy from a table.

if there is row-level duplicacy, we can assign a primary key to remove dupicants.

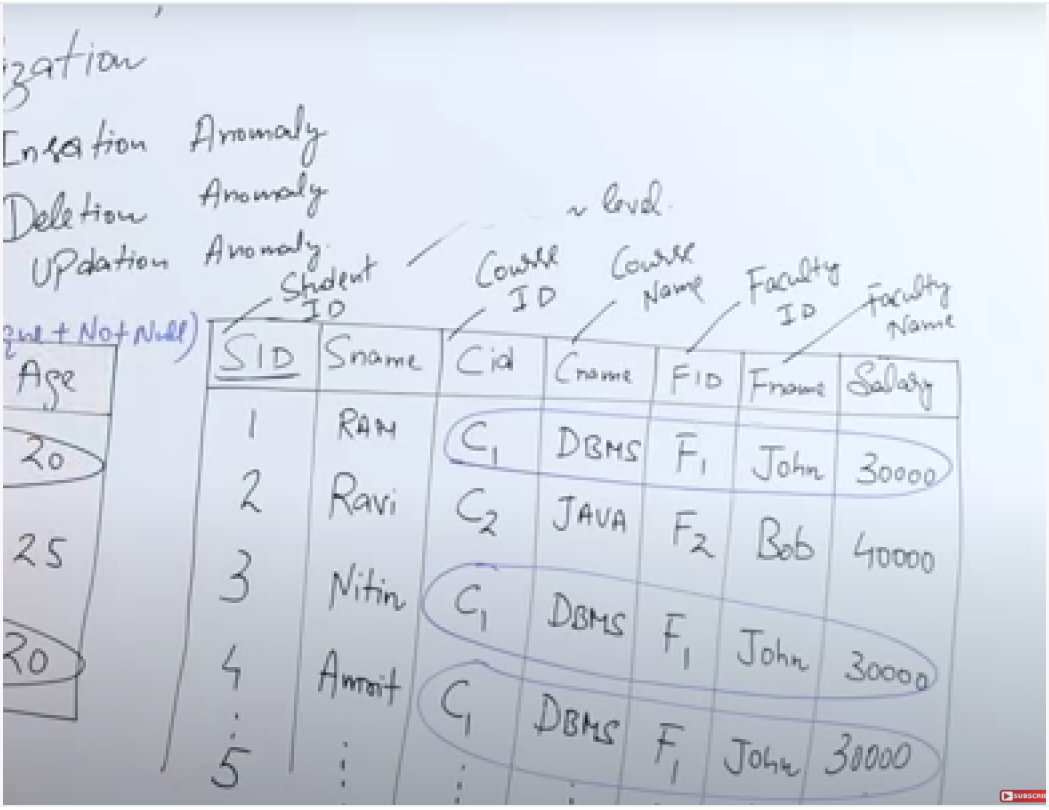
we will set a column as primary key and it will take care of the duplicate values.



in column level redundancy some columns can have the same entries.

there three type of problem can occur :

1. Insertion anomaly - problem for new entries
2. deletion anomaly - useful data may be lost
3. updation anomaly - many duplicate entires will be updated.

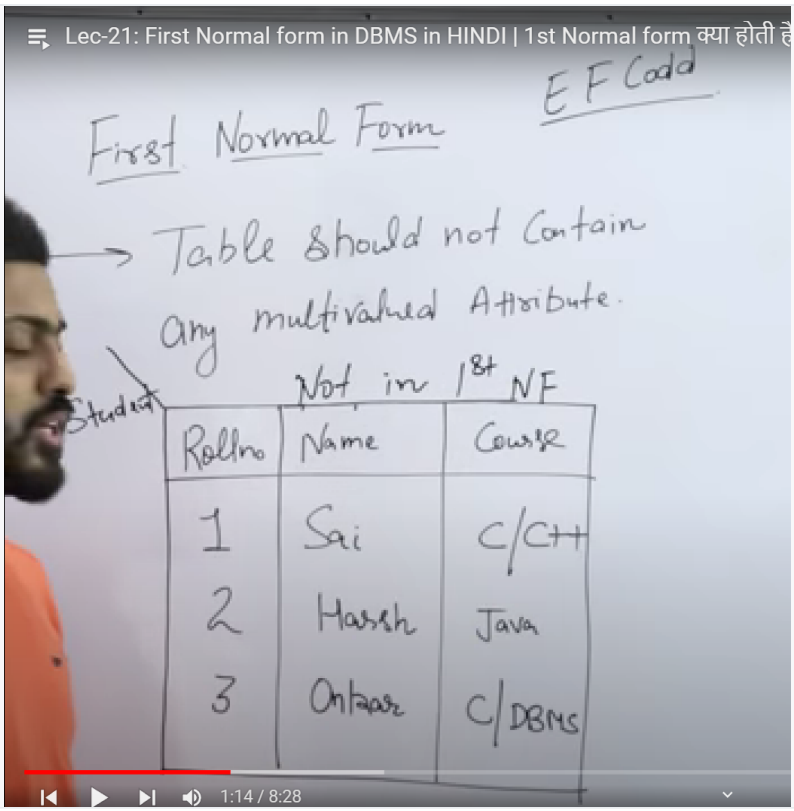


Normalization: decomposes the tables till the redundancy is removed.

1st Normal Form:

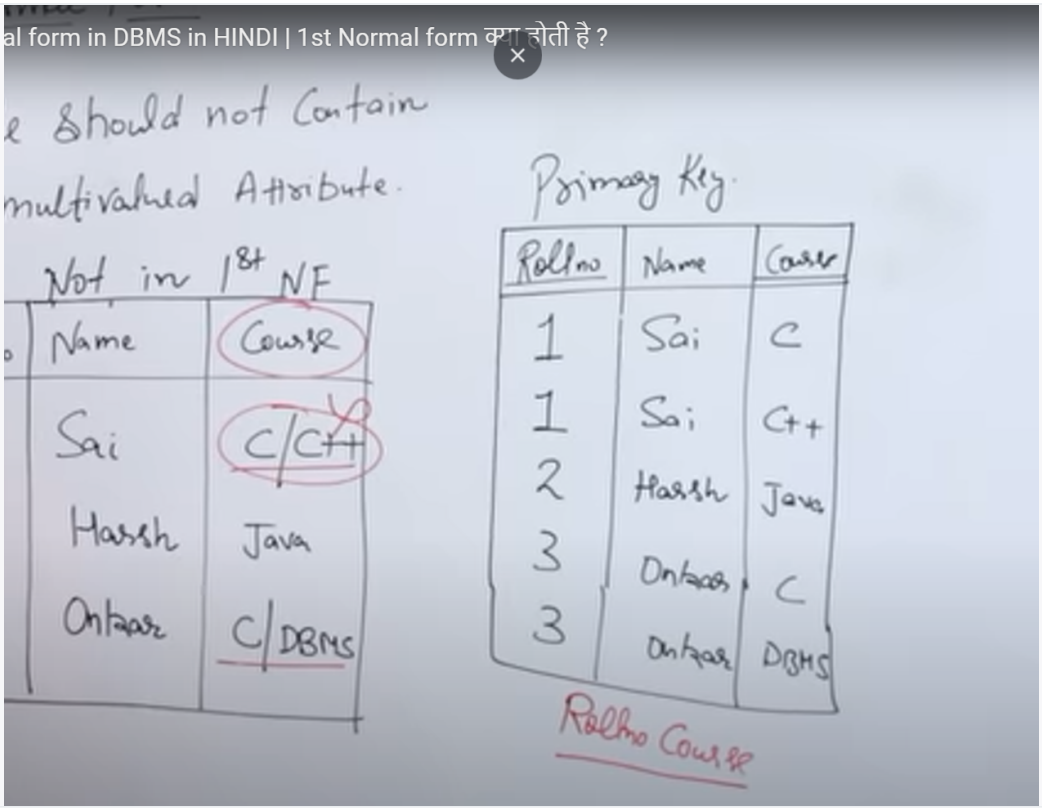
table should not contain any multivalued attribute.

like in below eg. 1 has course c/c++ that should not be the case.

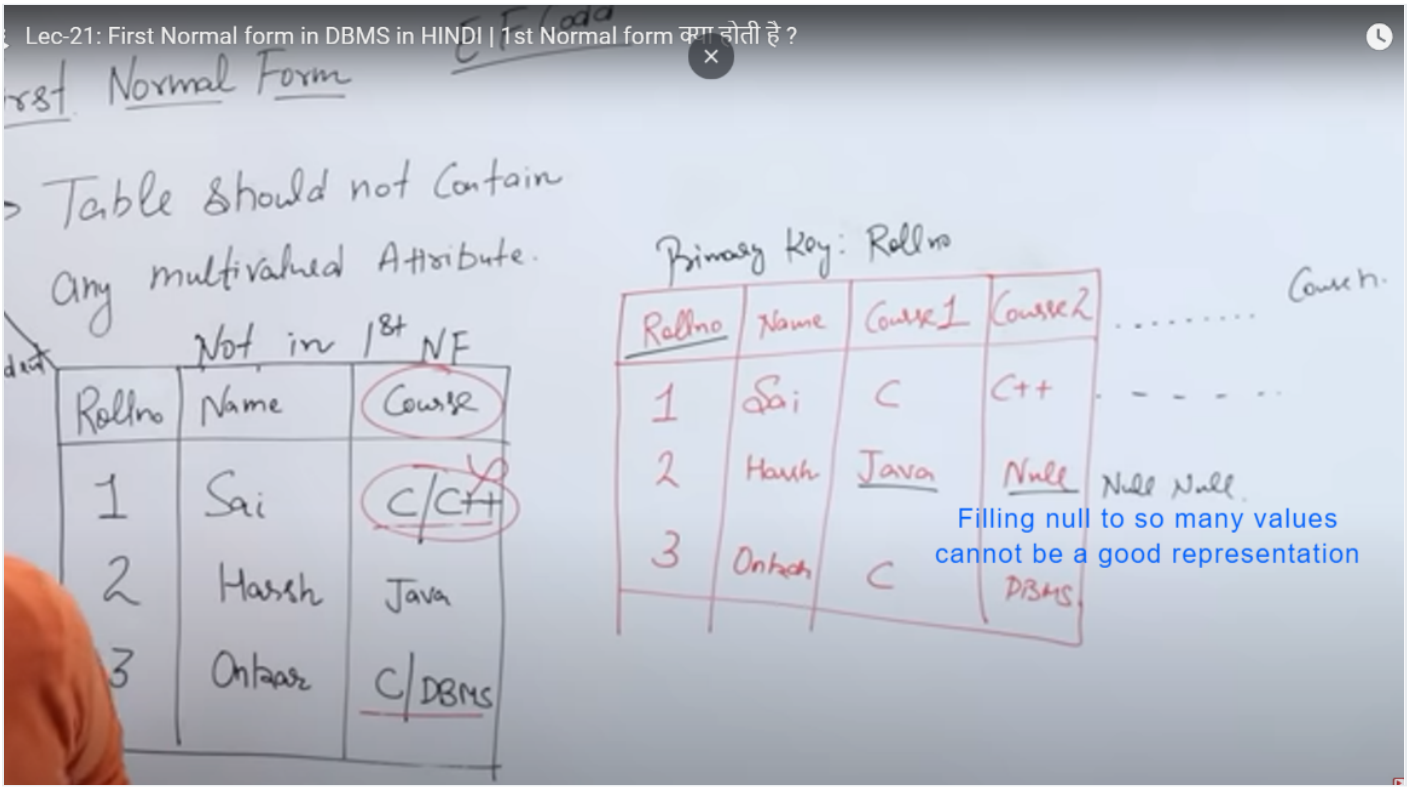


To convert a table to 1st normal form we have 3 solutions:

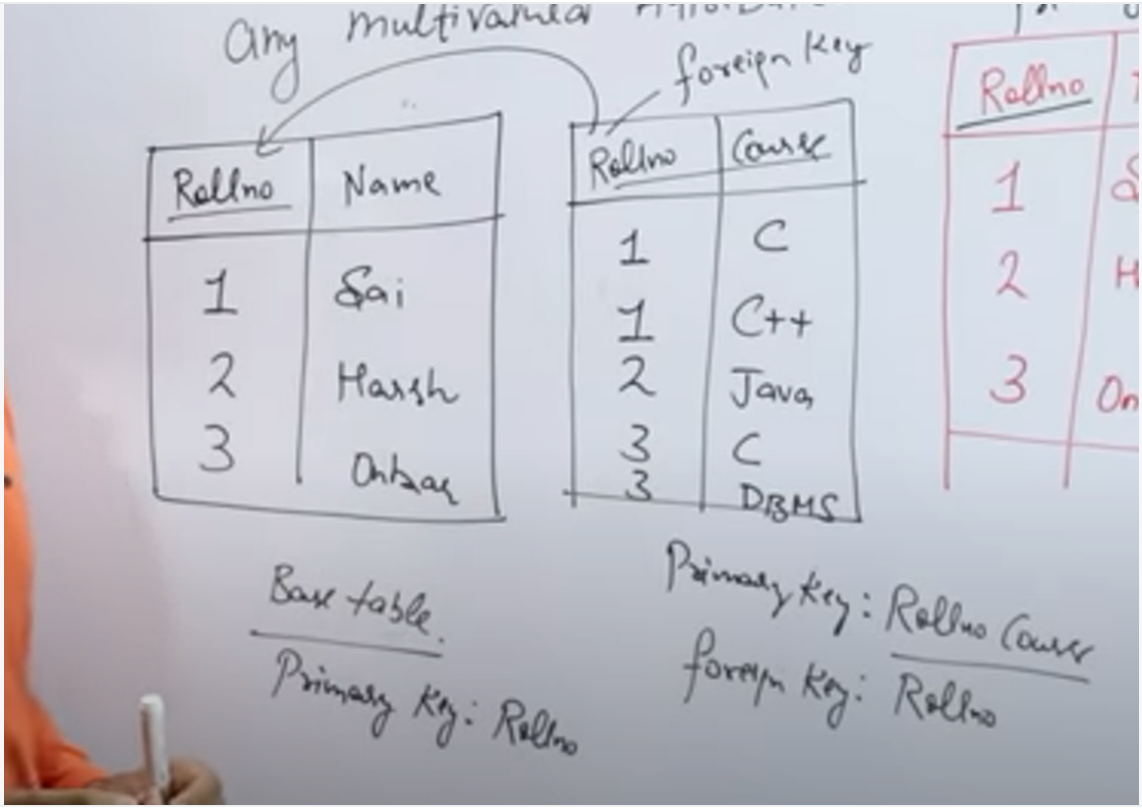
1. we can take the mutiple values as seperate entries like one entry as (1 sai c) and another as (1 sai c++) and we can make the primary key composite of (rollno and course).



2. we can take the maximum number of values an attribute is having and make that number of separate columns.



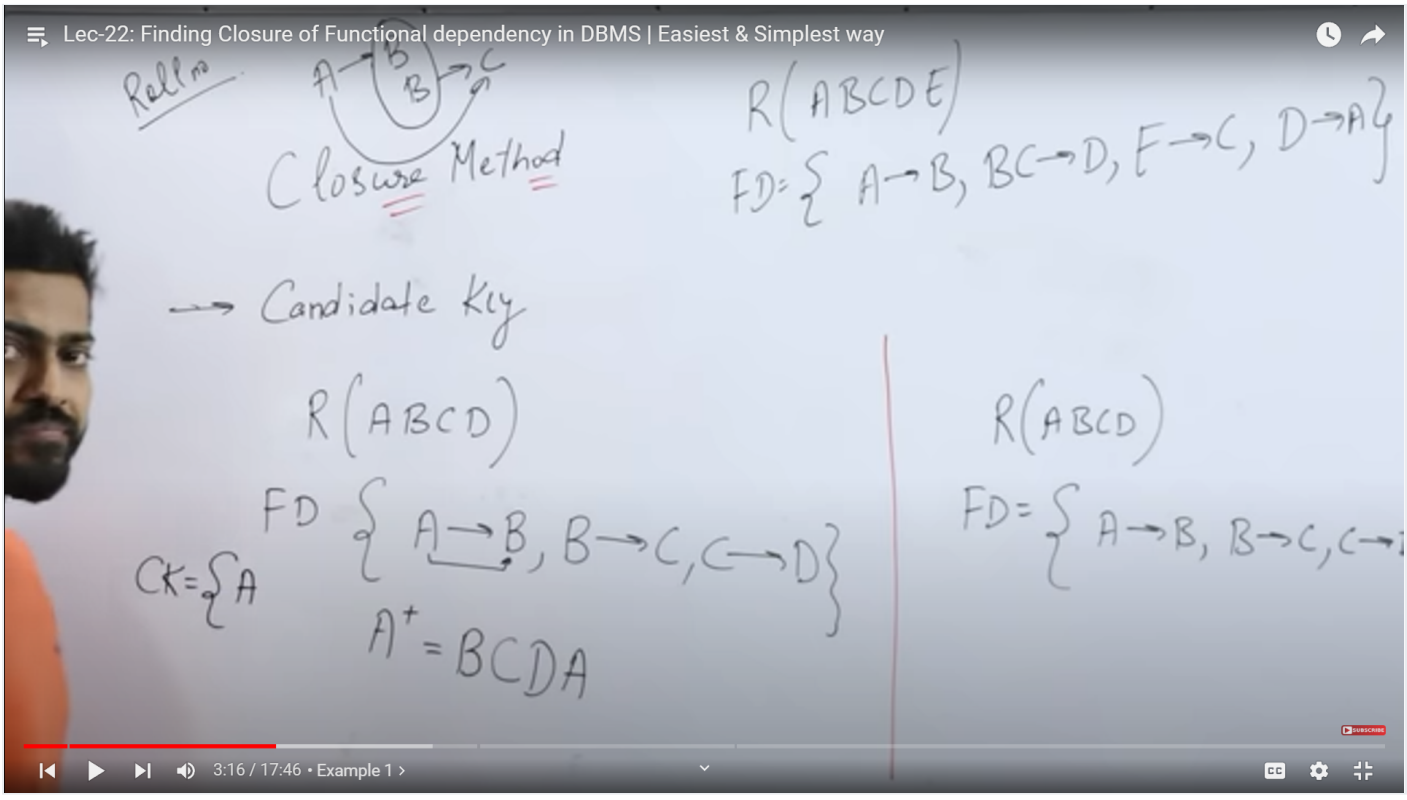
3. The table can be divided into two different tables and the primary key of the base table will be the foreign key of the referencing table.

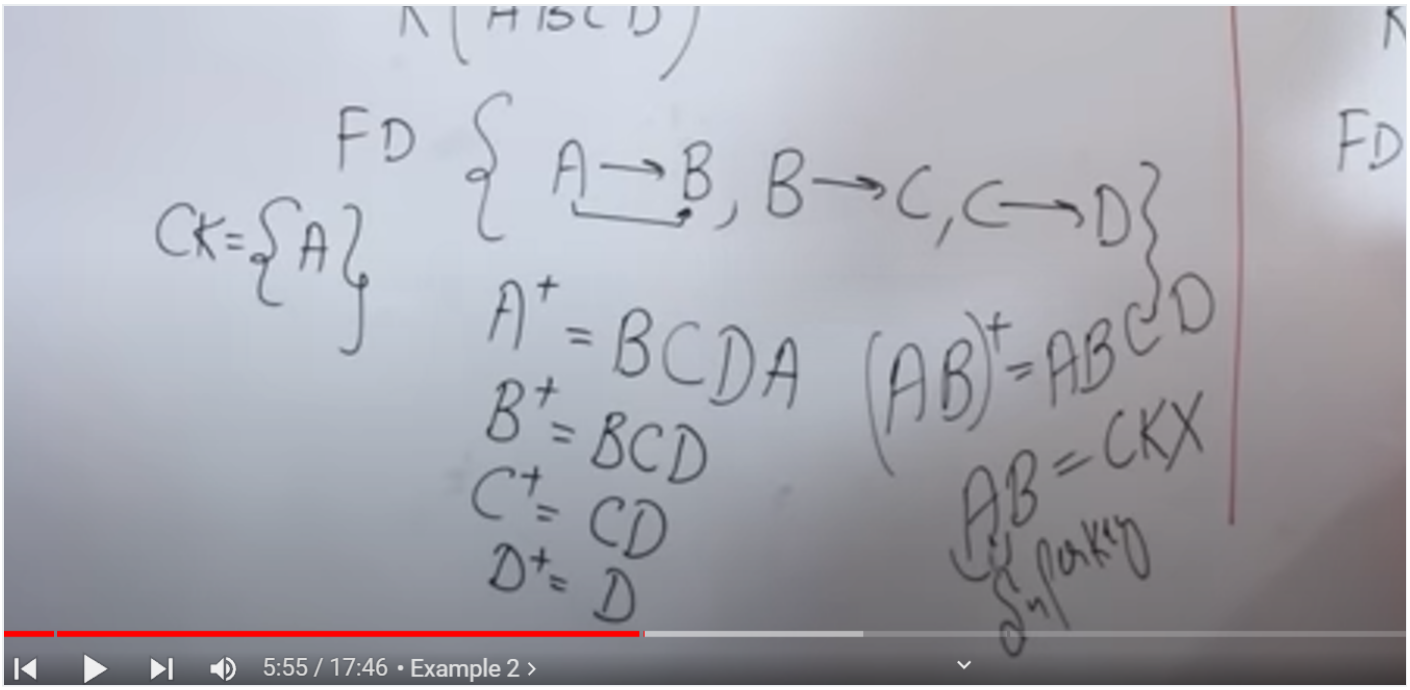


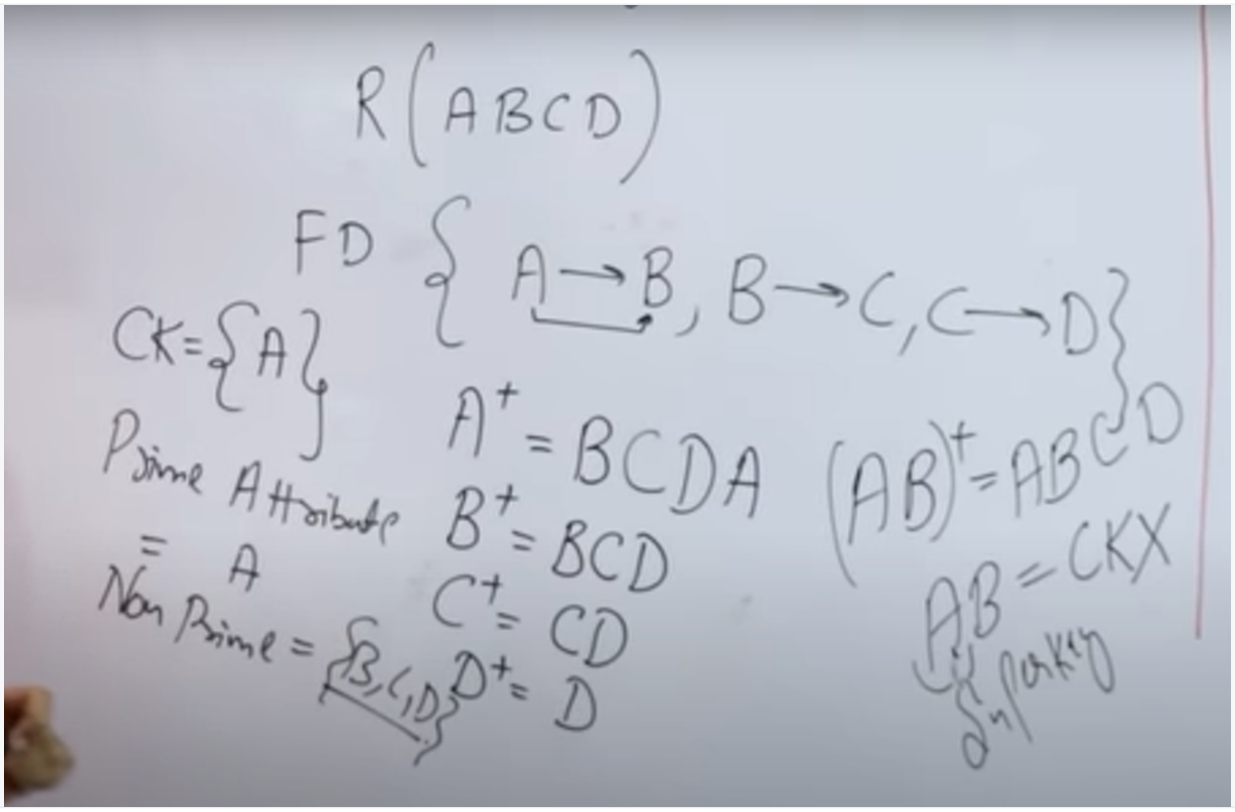
Closure Method

use to find the all candidate keys.

if the closure of an elemnet can determing all the elemnts of table it is a candidate key.

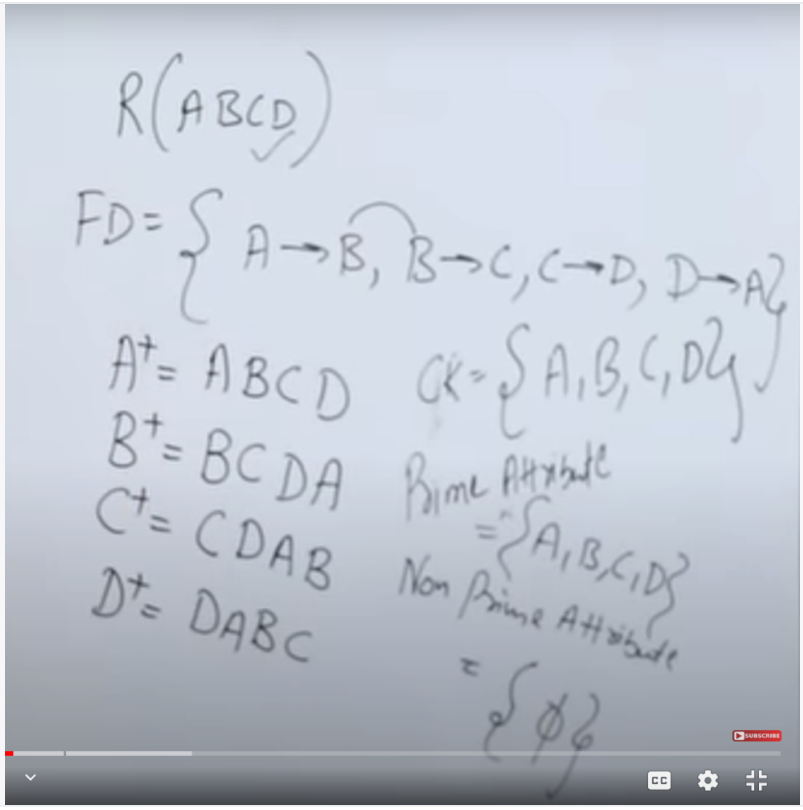






here AB combine is not a candidate key although it determines all the elements of the relation, it is combinly determining. so, it's not a candidate key it's a super key. a candidate key is a minimalist key.

prime attribute: it is the attribute from relation which is used in making the candidate key.



Functional dependencies:

they are of two types:

1. trivial FD: Valid in all cases.

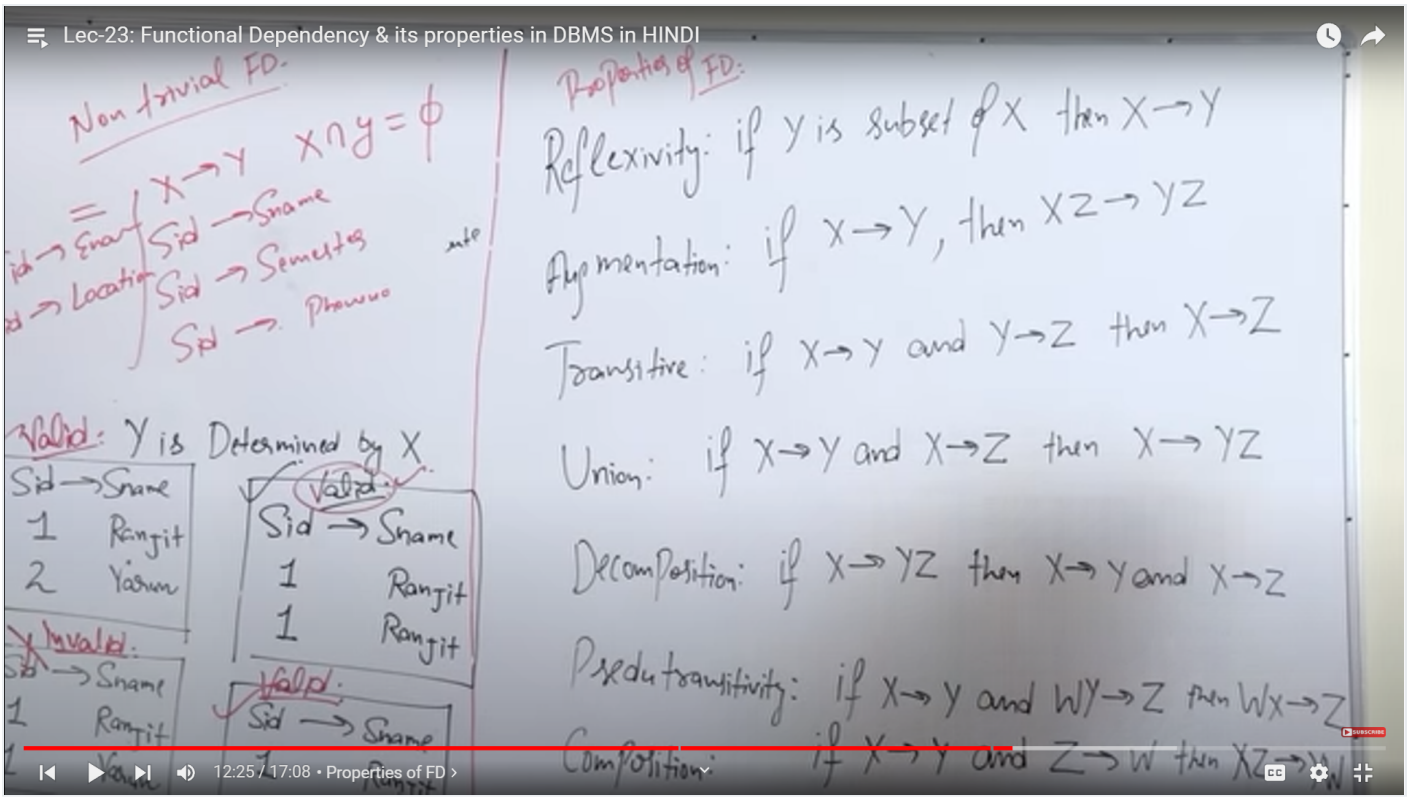
   if x→y (x deteremines y) then y is a subset of x.

   and if we take the intersection of LHS n RHS will never be NULL. (x n y != NULL)

2. non-trivial FD: if x→ y then y is not a subset of x

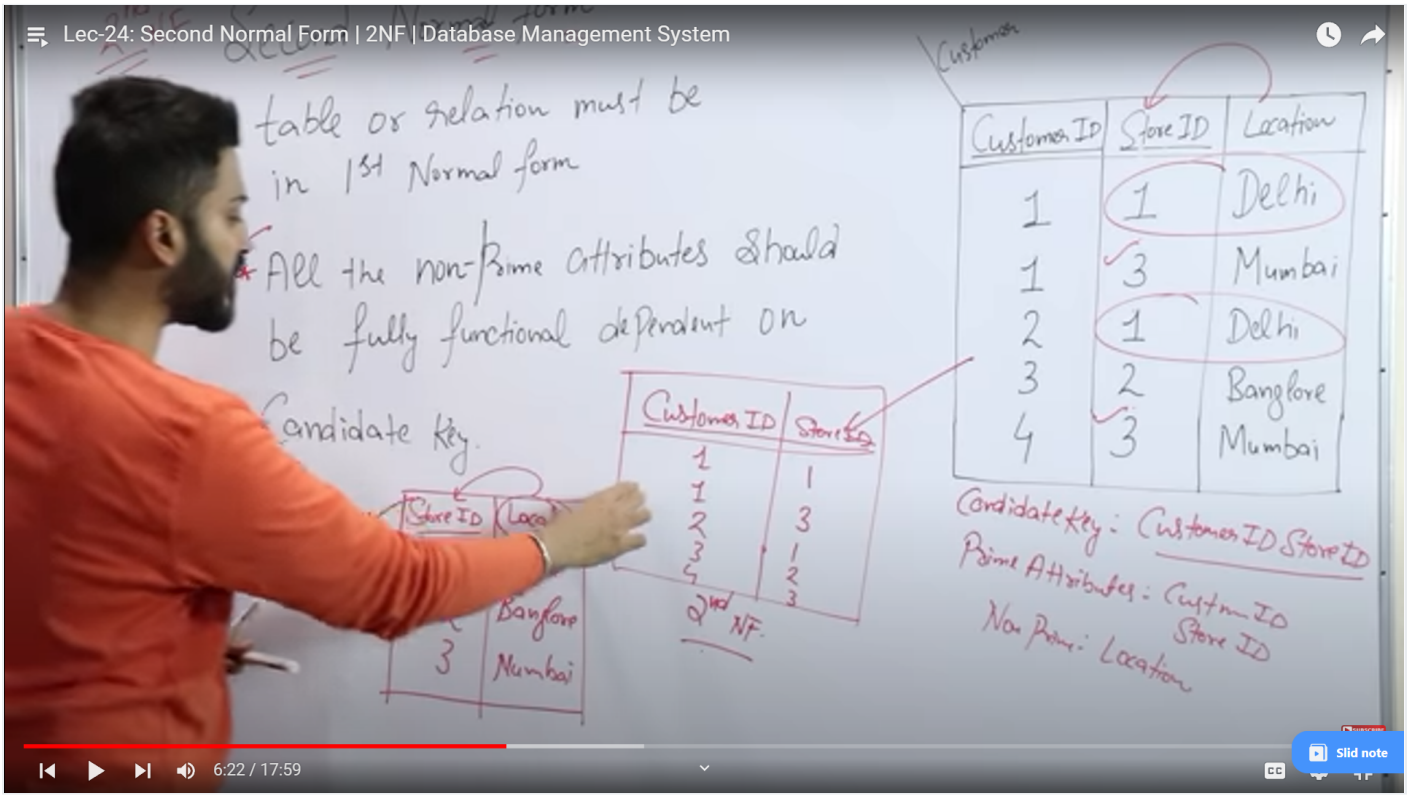
   and their intersection is always null.

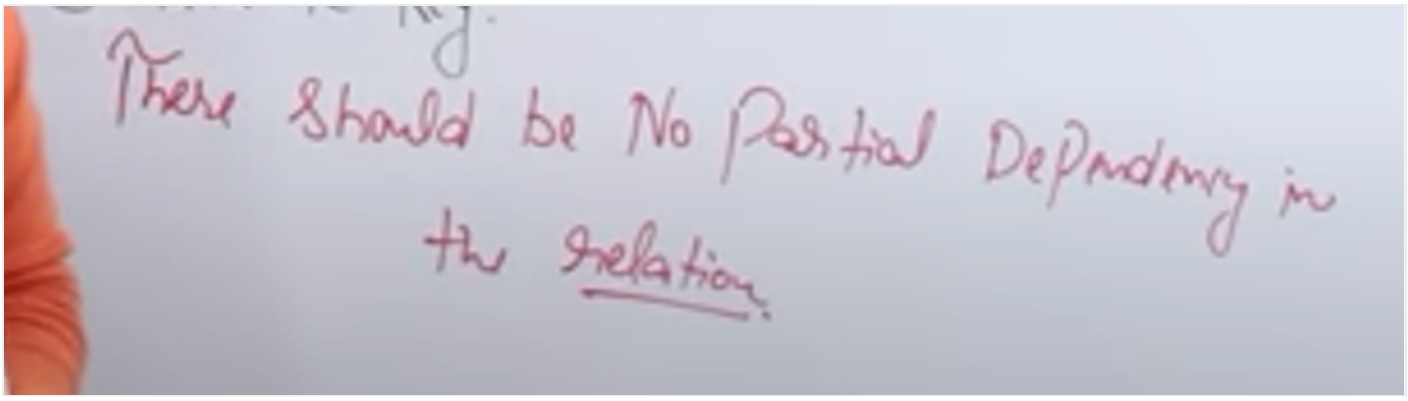
y is dependent and x determinant.



whenever y has any confusion it will go to determinant. that whether it is valid or not.

2nd Normal Form:

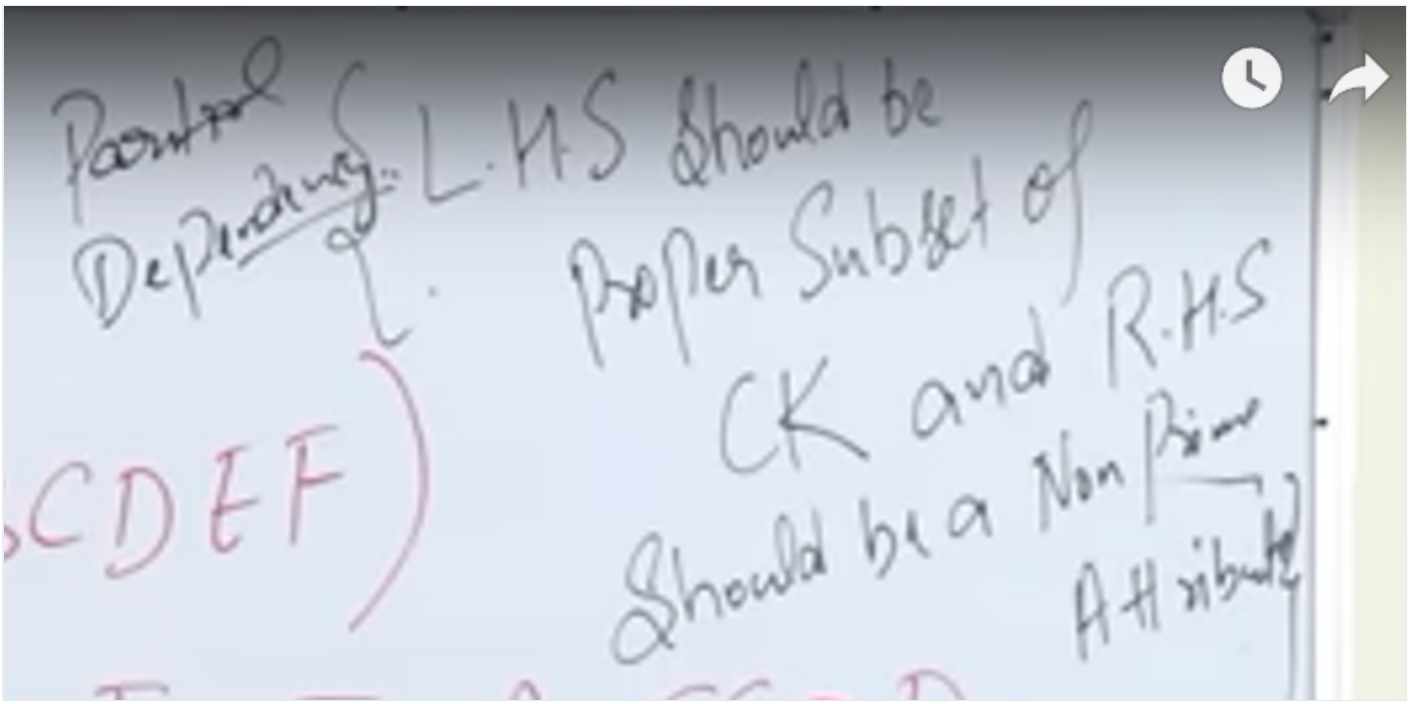




if any non-prime attribute is determined by the part of the candidate key (for eg AB and B→C and C is non-prime, so C is determined by B, this is called partial dependency which should not be present).

if this condition appears:

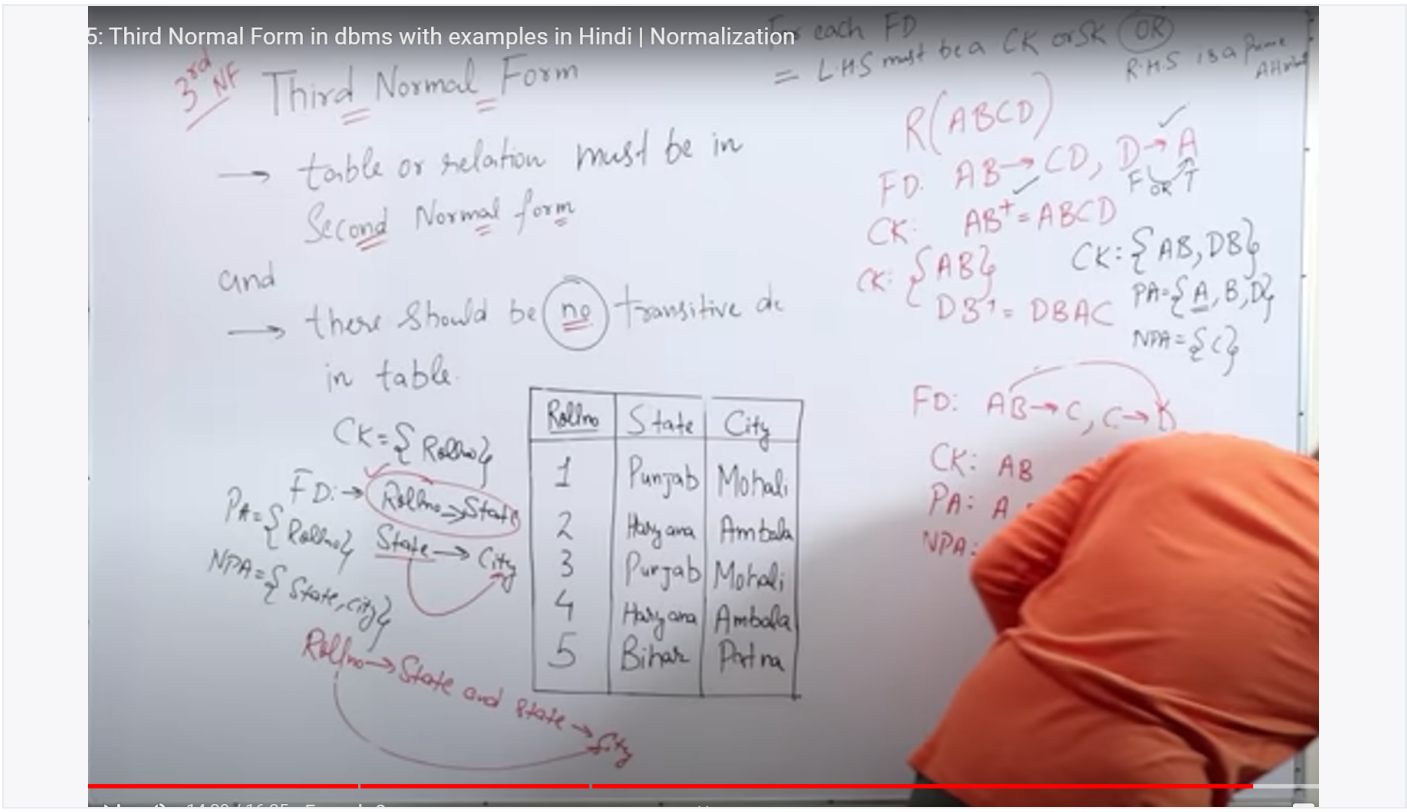
LHS is a proper subset of CK and RHS is a non prime attribute.



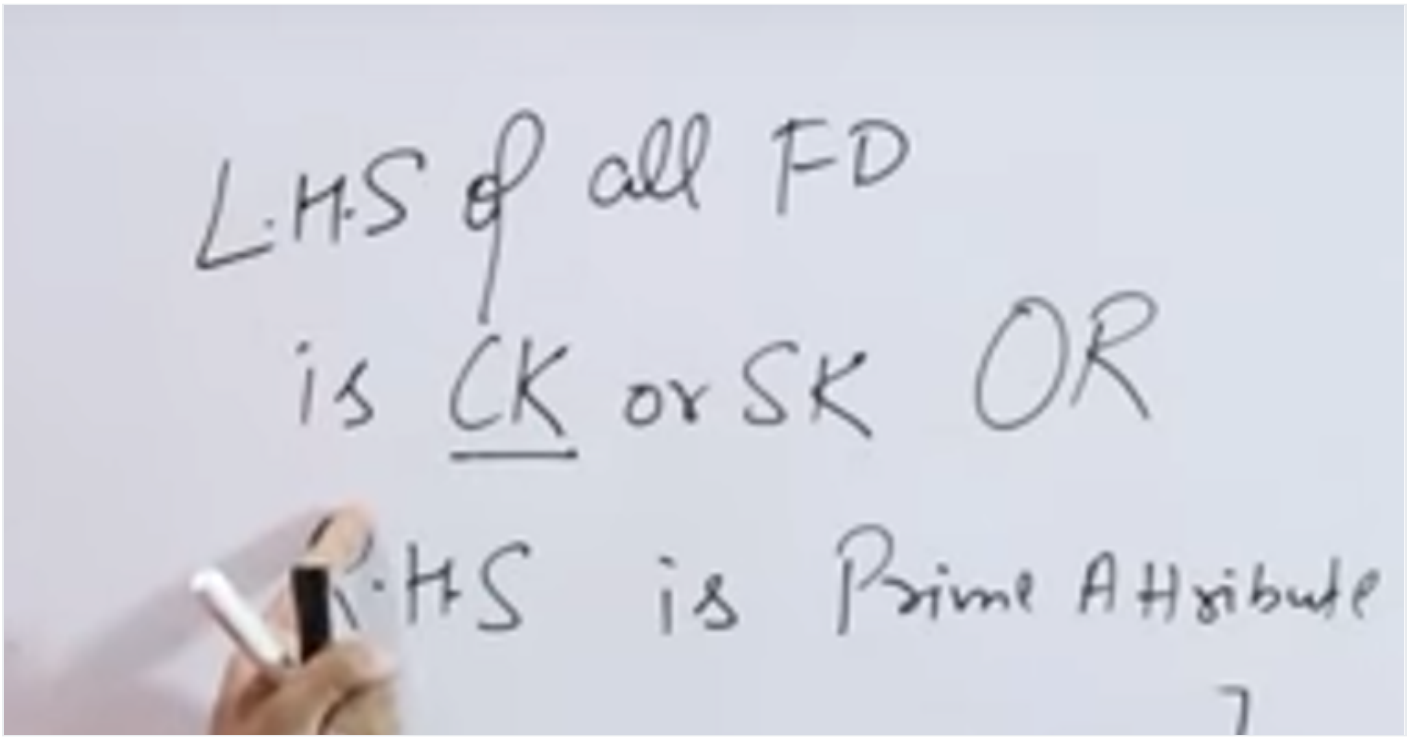
then the table is not in the 2nd normal form.

3rd Normal Form

a non prime attribute cannot be determined by a non prime attribute.



condition for a table to be in 3rd NF:



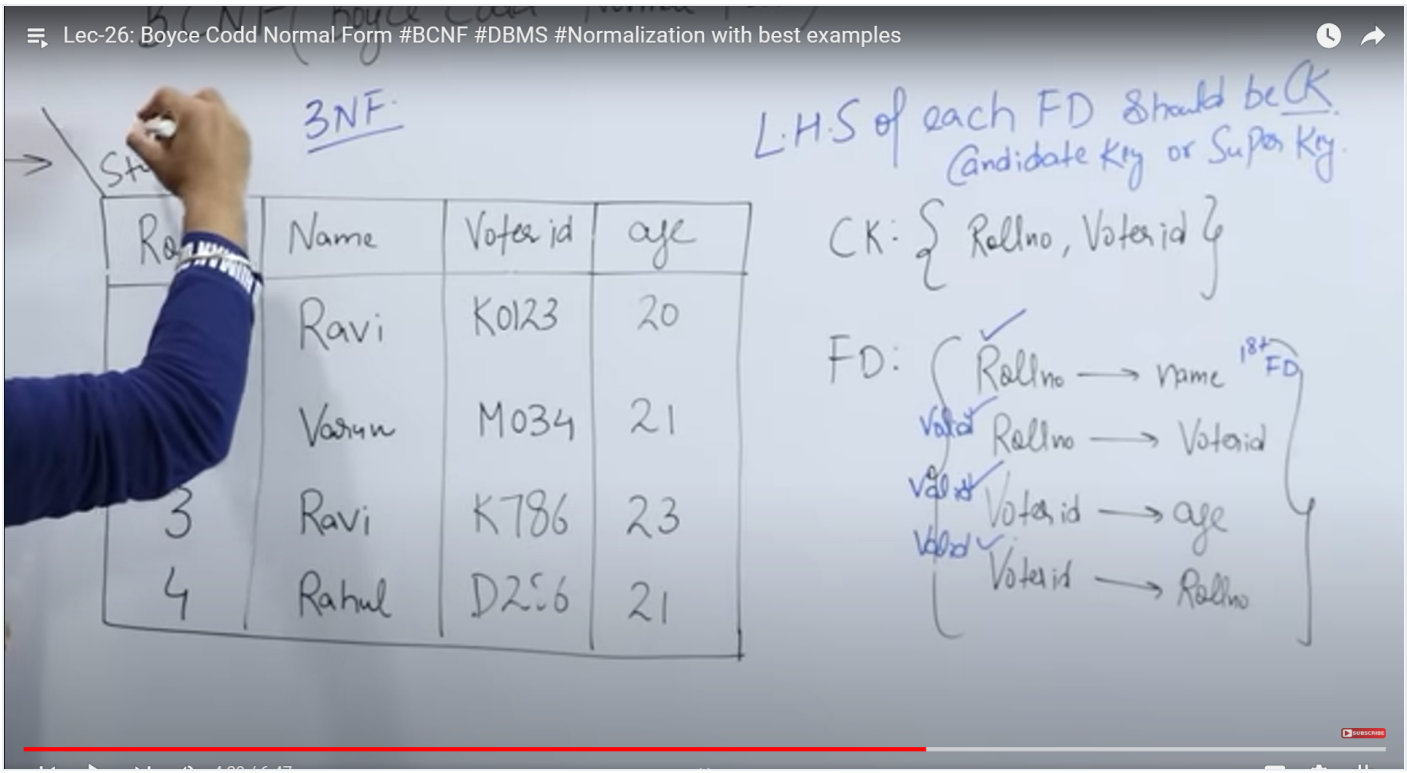
Boyce codd normal form (BCNF)

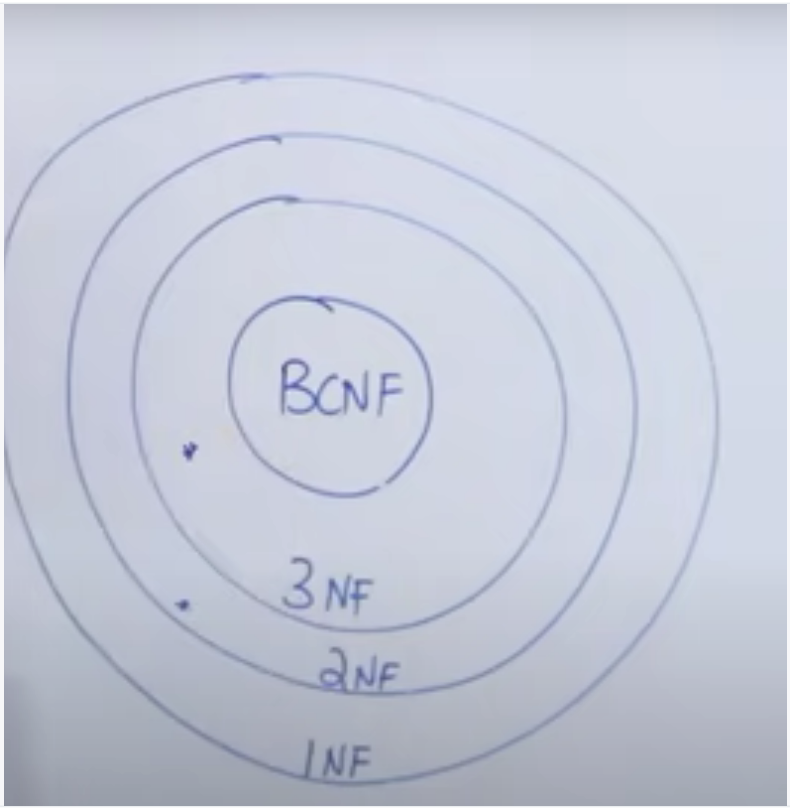
a special case of 3rd normal form.

BCNF always have lossless decomposition.

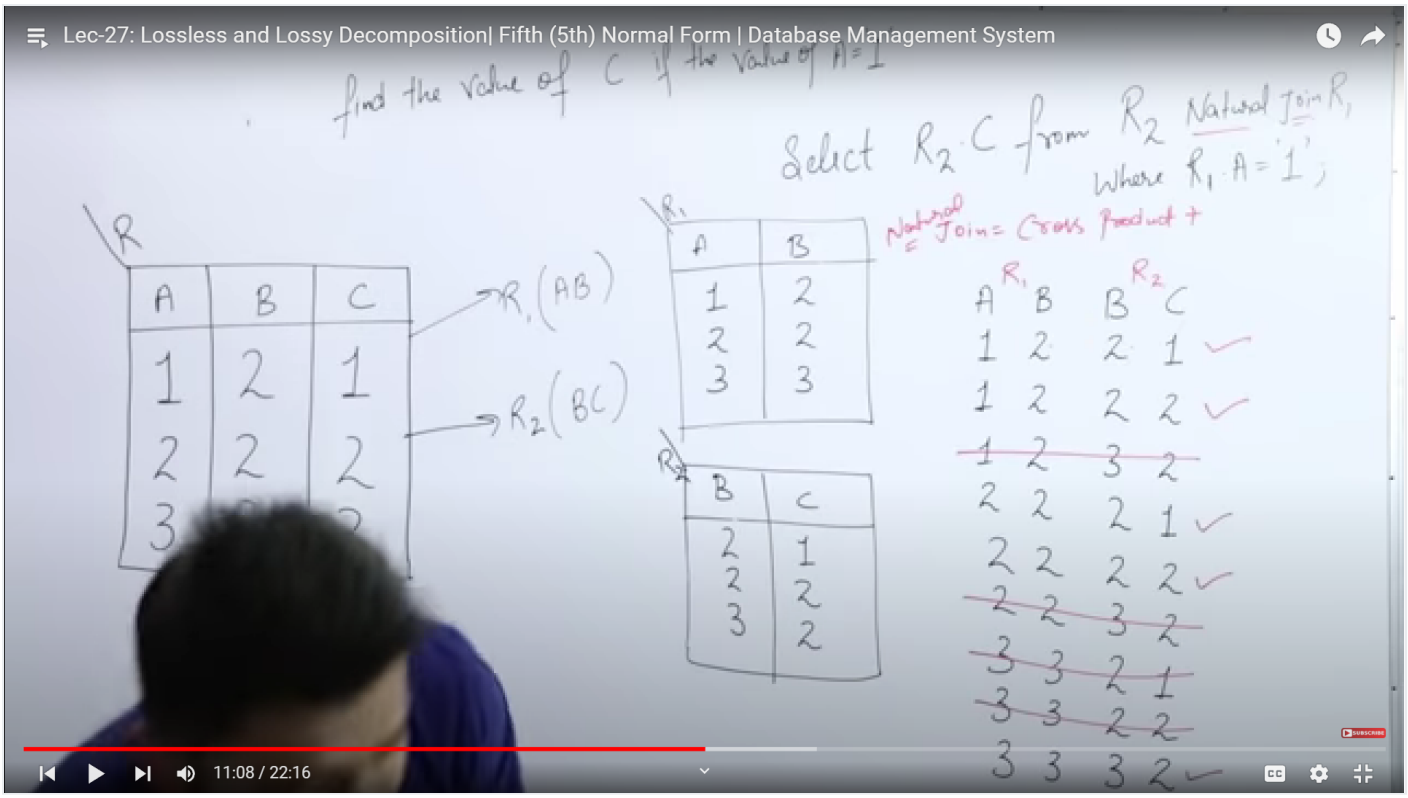
Condition: In LHS of all function dependency, candidate key should be present.

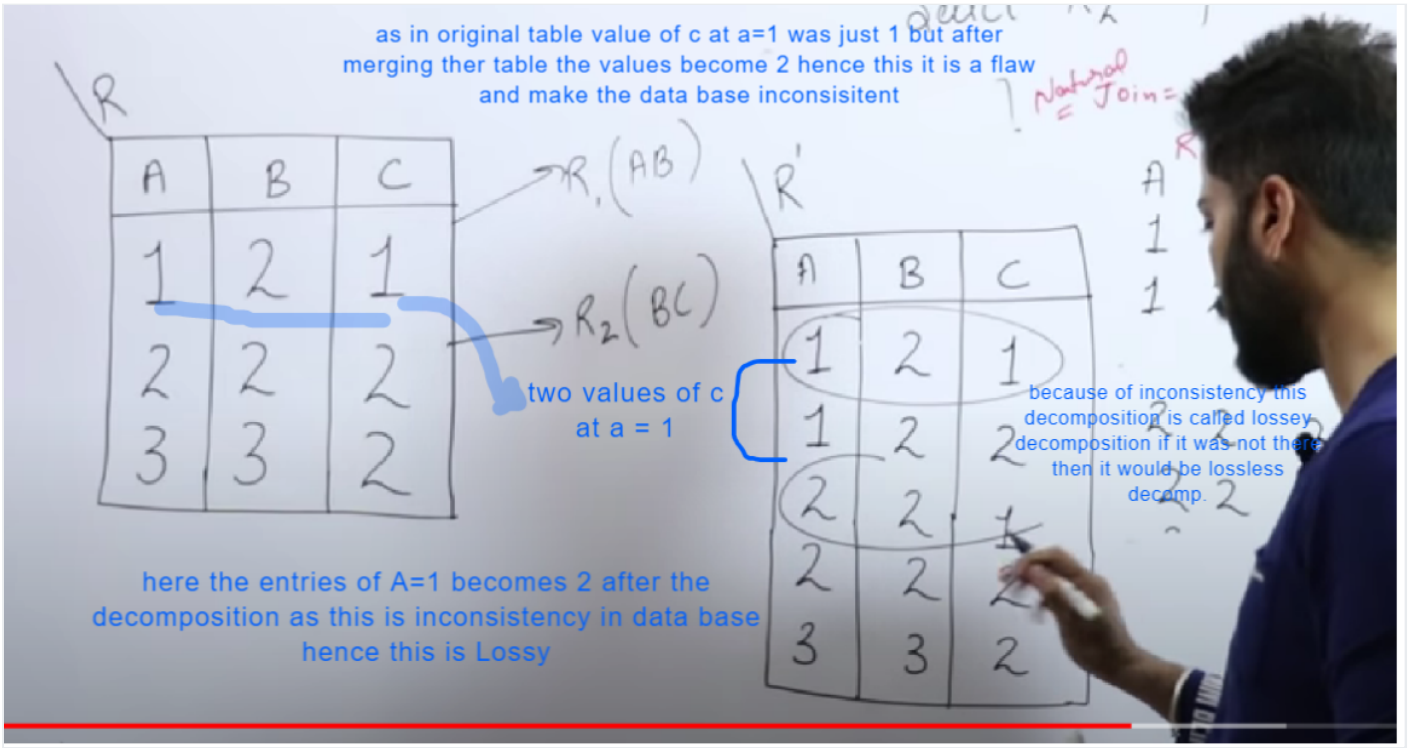
⇒ Table should be in 3rd Normal form

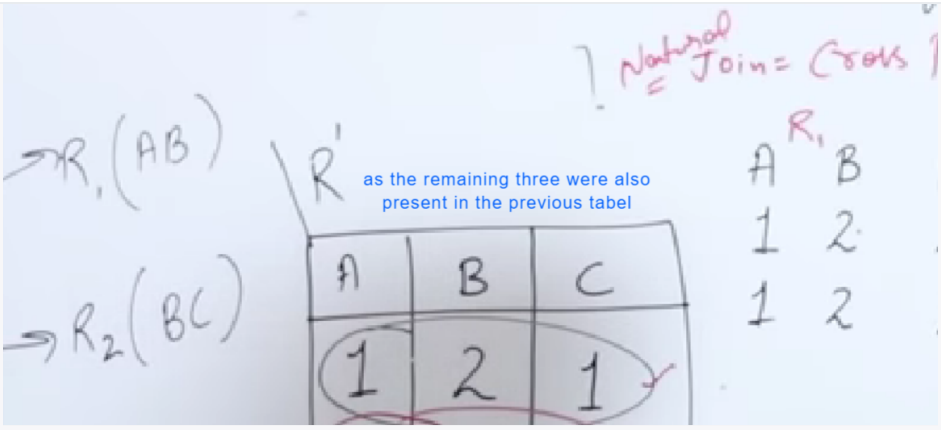




Lossy vs lossless Decomposition

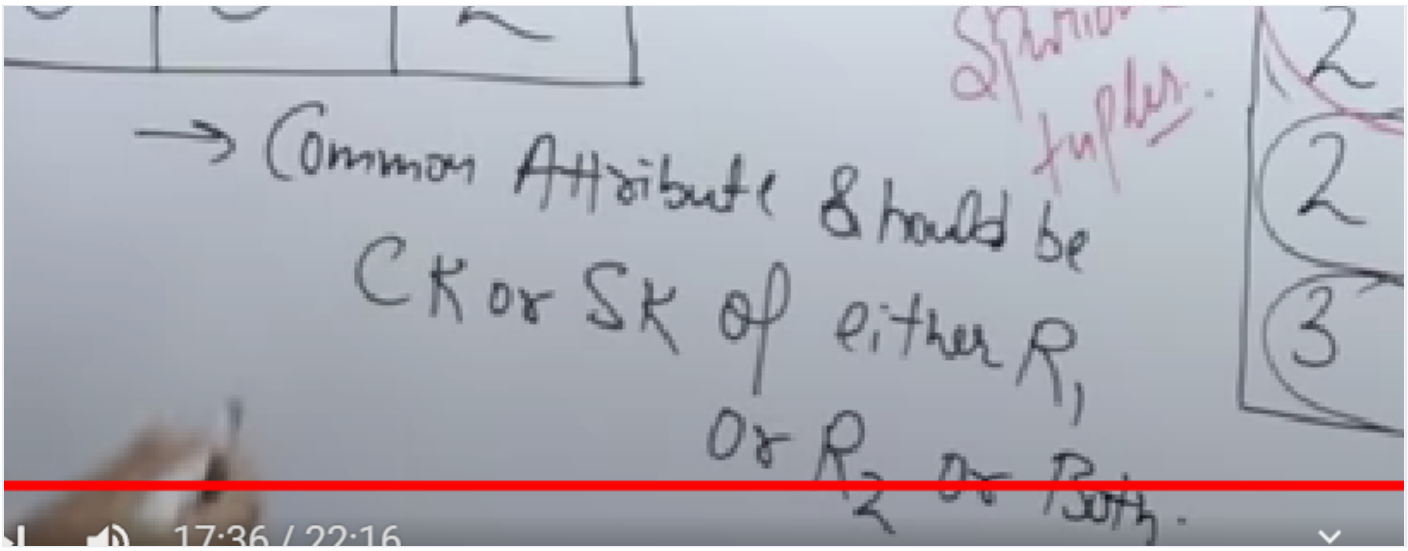

Lec-27: Lossless and Lossy Decomposition| Fifth (5th) Normal Form | Database Management System
find the value of C if the value of A=1
Select R2.C from R2 Natural Join R1 Where R1.A = 1;
R
A B C
1 2 1
2 2 2
3 3 2
R1(AB)
R2(BC)
R1
A B
1 2
2 2
3 3
R2
B C
2 1
2 2
3 2
Natural Join = Cross Product +
A B B C
1 2 2 1
1 2 2 2
1 2 3 2
2 2 2 1
2 2 2 2
2 2 3 2
3 3 2 1
3 3 2 2
3 3 3 2
11:08 / 22:16


as in original table value of c at a=1 was just 1 but after merging ther table the values become 2 hence this it is a flaw and make the data base inconsisitent
R
A B C
1 2 1
2 2 2
3 3 2
R1(AB)
R2(BC)
two values of c at a = 1
R'
A B C
1 2 1
1 2 2
2 2 1
2 2 2
3 3 2
because of inconsistency this decomposition is called lossey decomposition if it was not there then it would be lossless decomp.
here the entries of A=1 becomes 2 after the decomposition as this is inconsistency in data base hence this is Lossy



when we decompose a table into other tables we put a attribute common in both tables, we should follow criteria otherwise inconsistency may occur:

⇒the common attribute between the split tables should be candidate key or super key of either R1 or R2 or both.



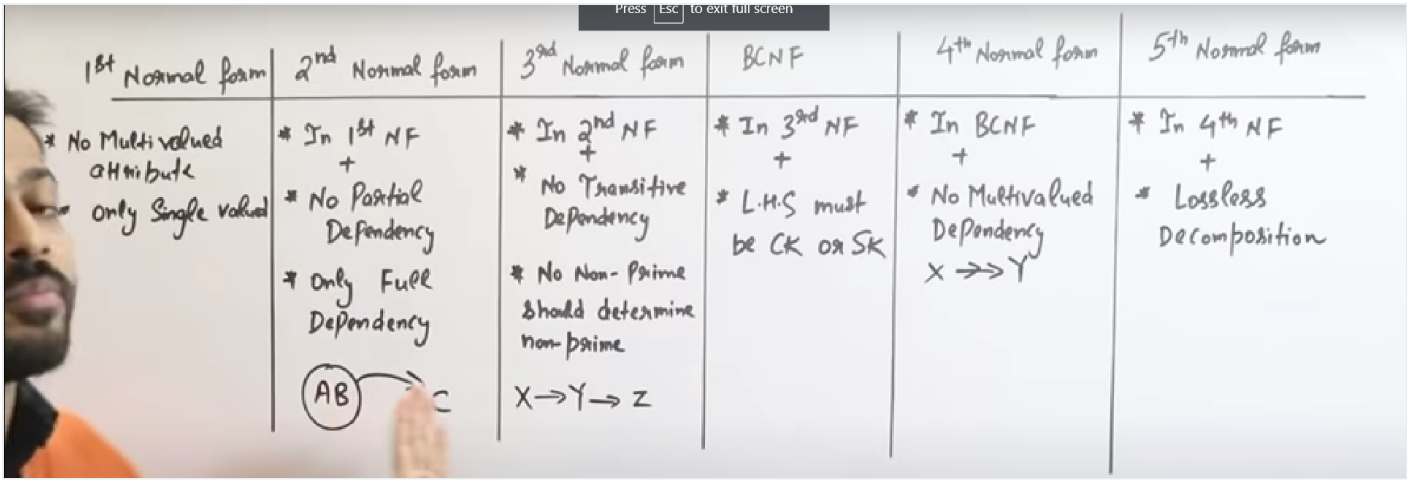
then we will get a lossless relation.

condition for lossless relation :-

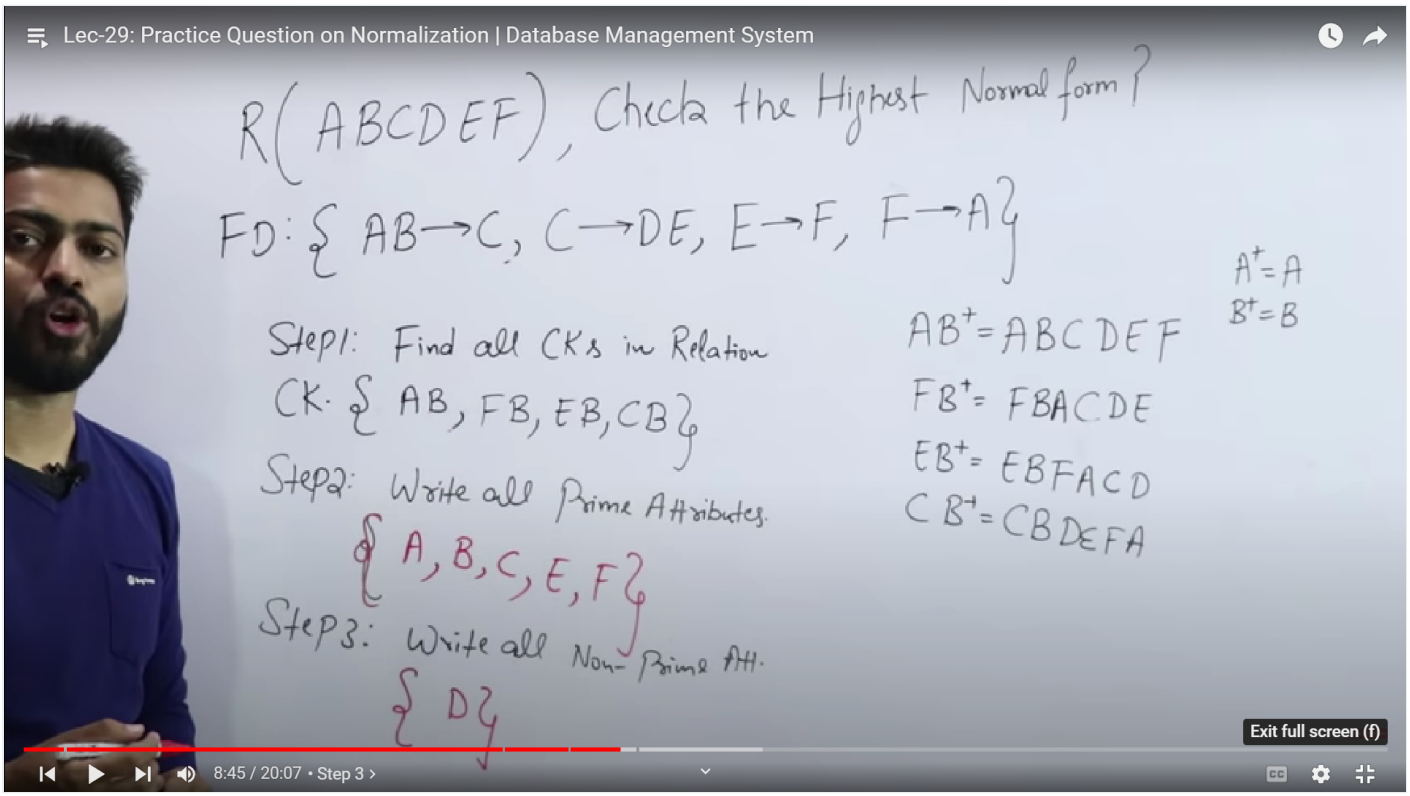
1. R1 U R2 = R (R is the original relation).
2. when we take the intersection of the split tables it should not be null

   → R1 n R2 != NULL
3. the common attribute should be C.Key/S.Key of R1 or C.Key/S.Key of R2 or both

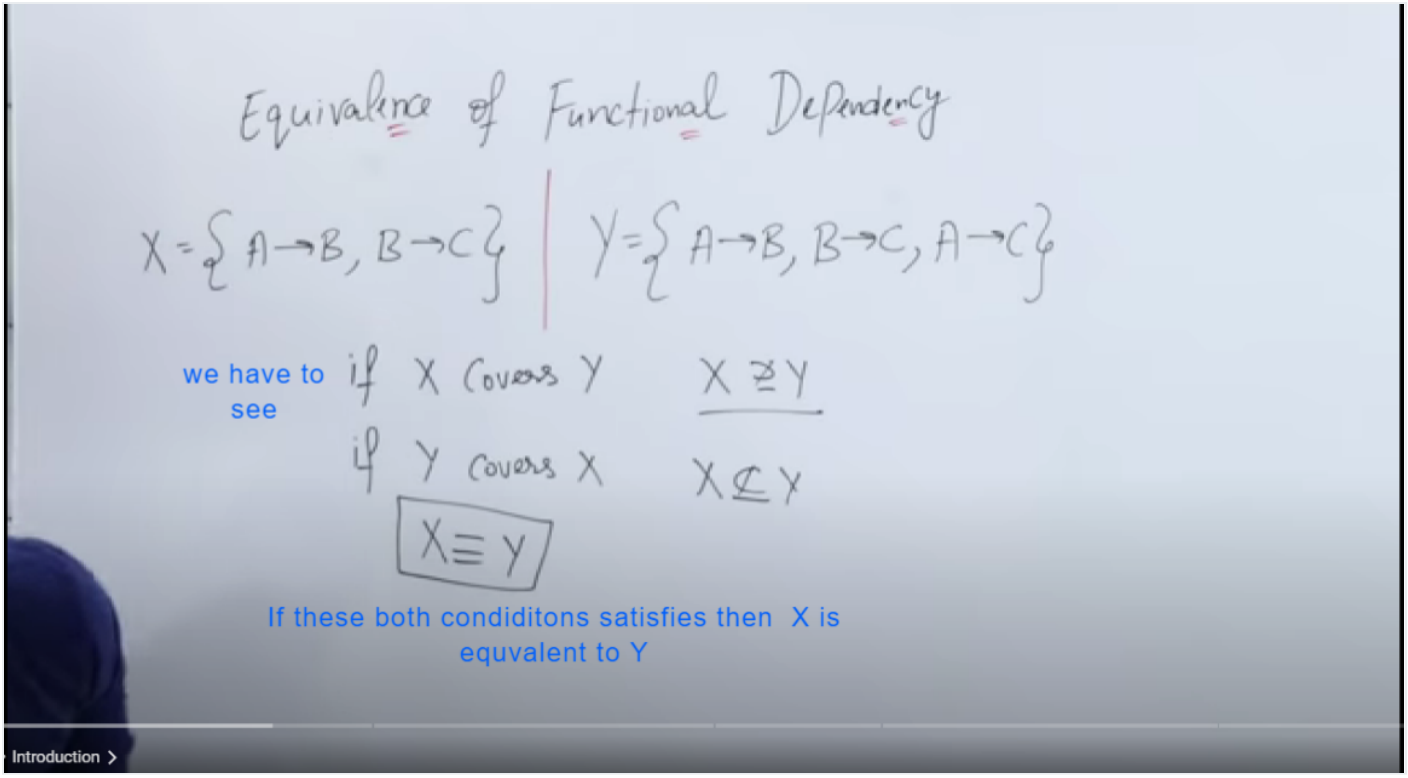
Summary of all normal form:



Practise Question-



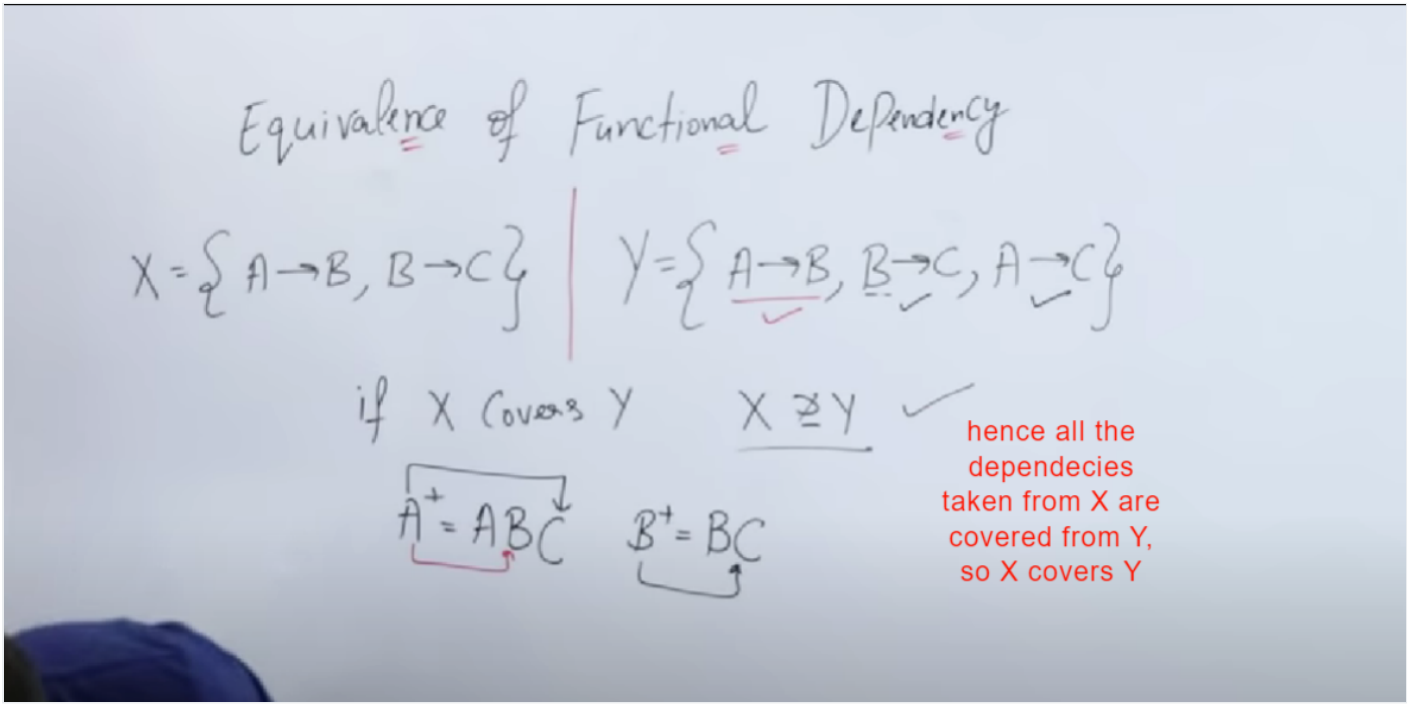
Cover and equivalence of function dependency

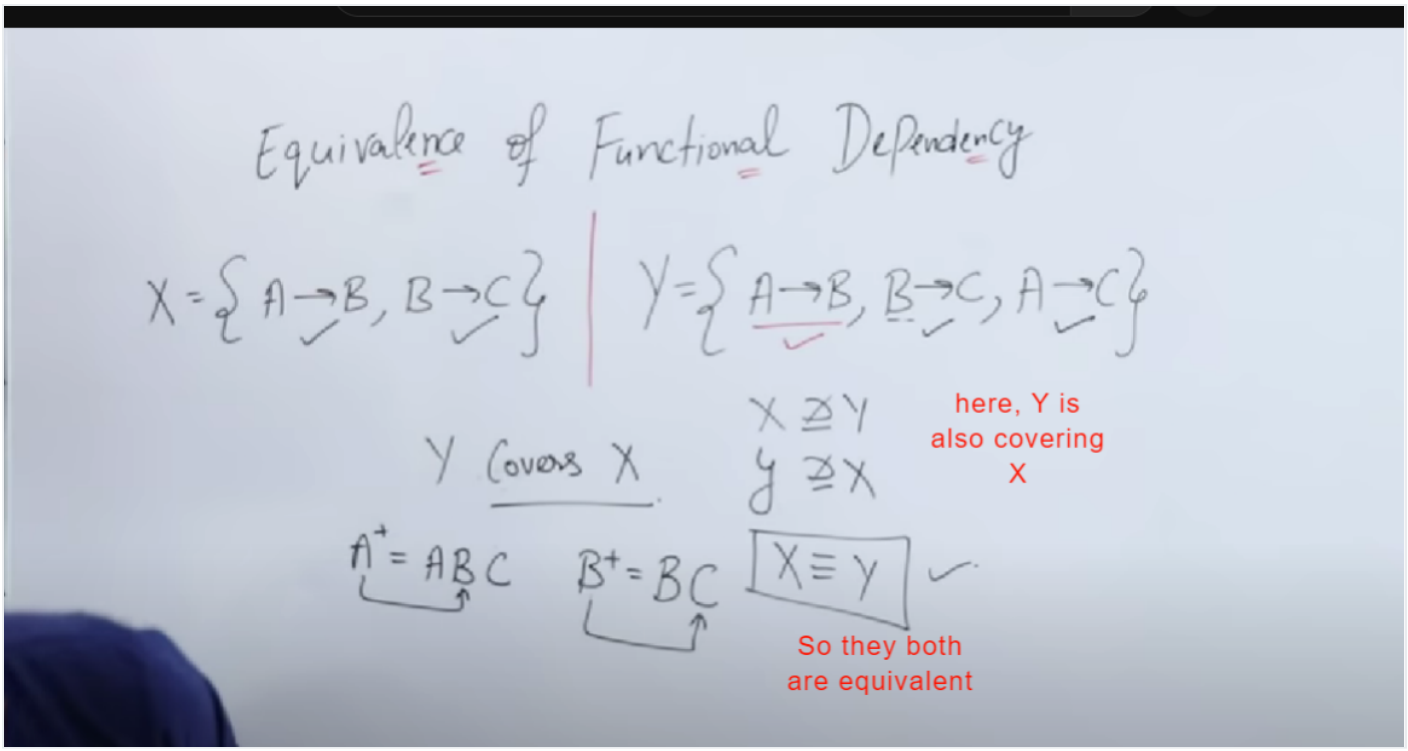


if the given check does y cover x, then take the RHS that is x and now take the LHS that is its functional dependencies and check it.

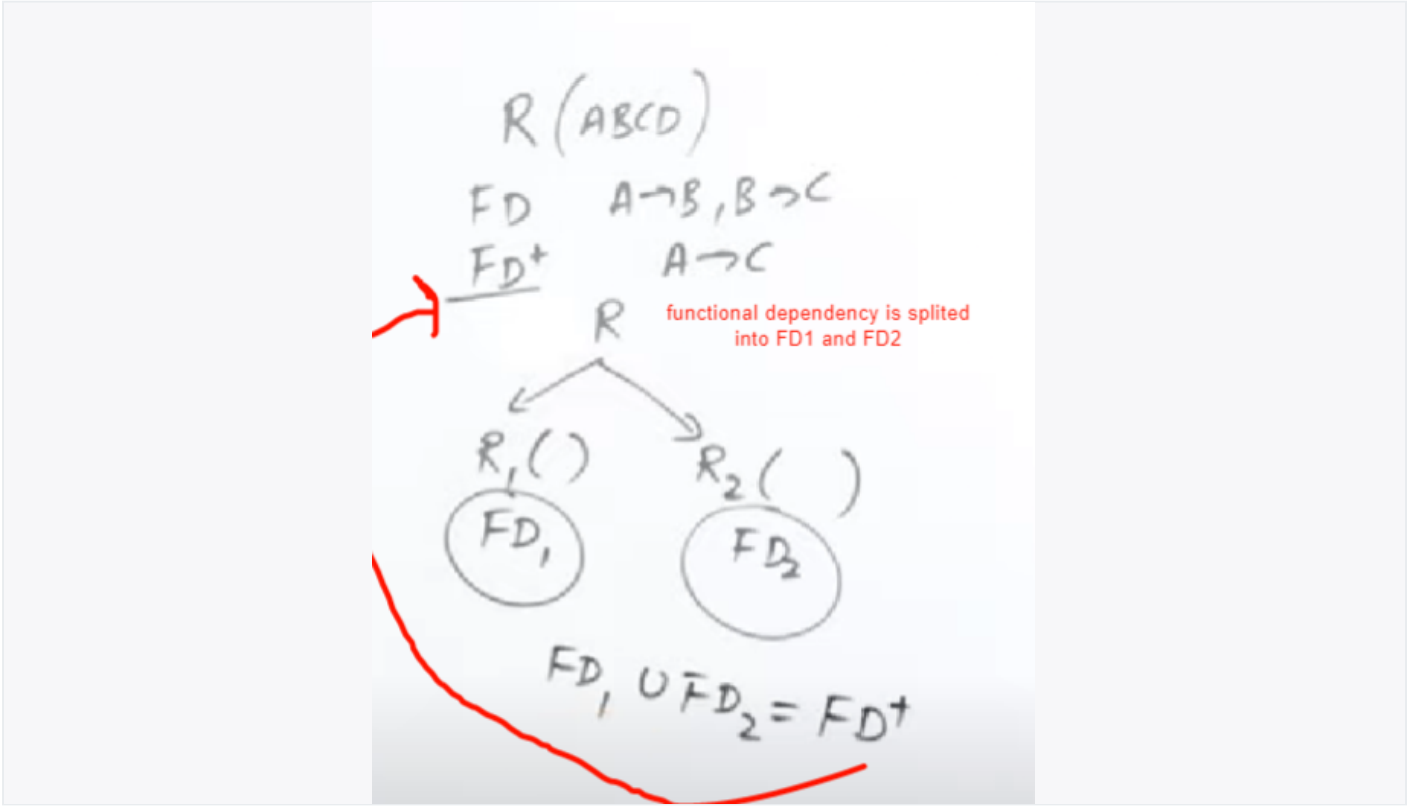
Now we will take the element from Y and take closure of it from X.

and then look for the all dependencies are covered in closure from Y taken from X.

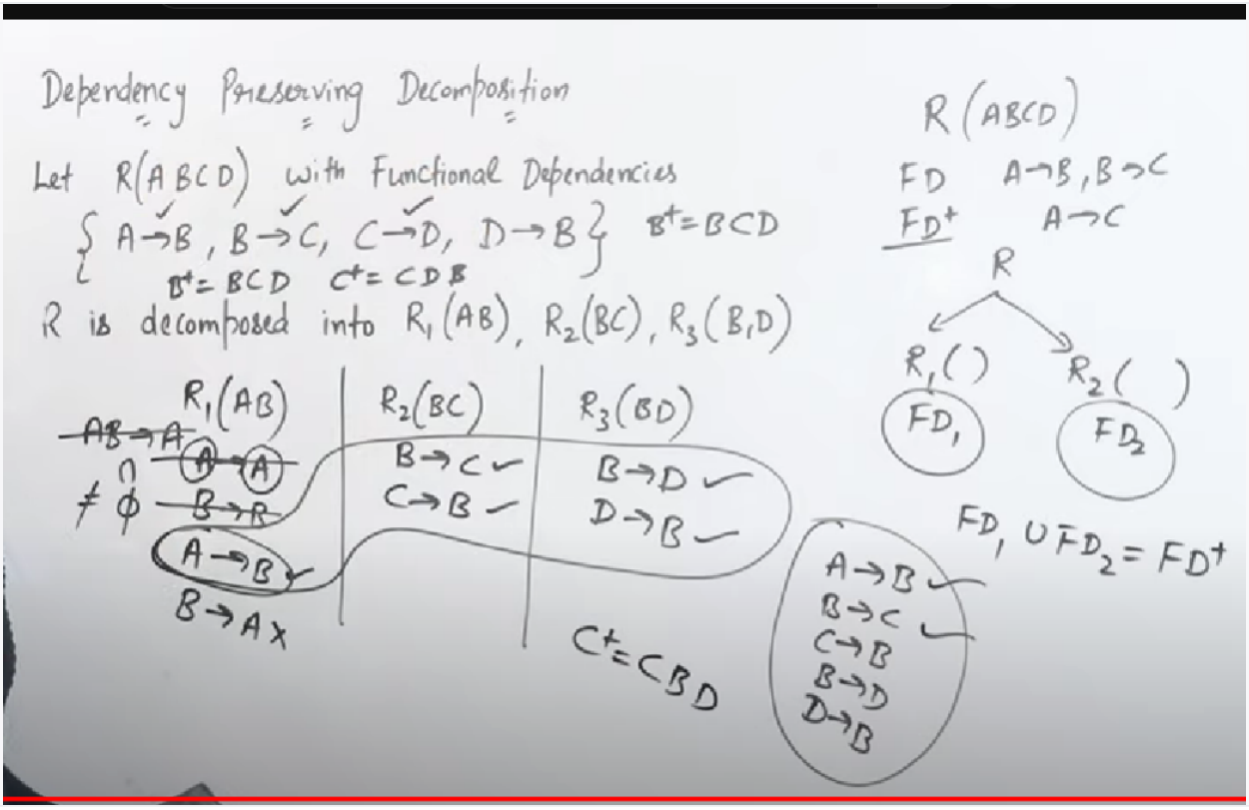




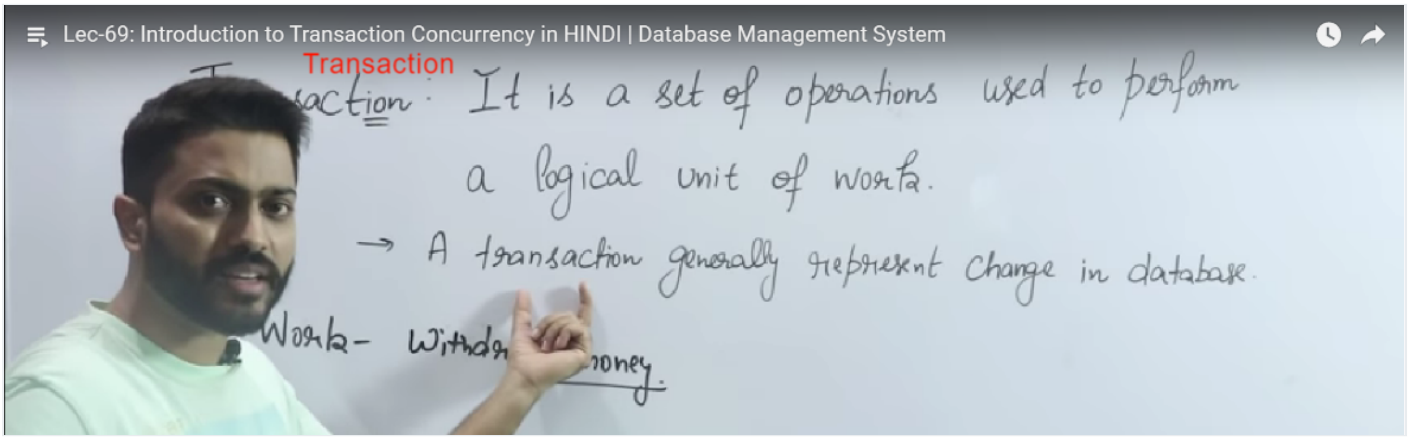
Dependency Preserving Decomposition

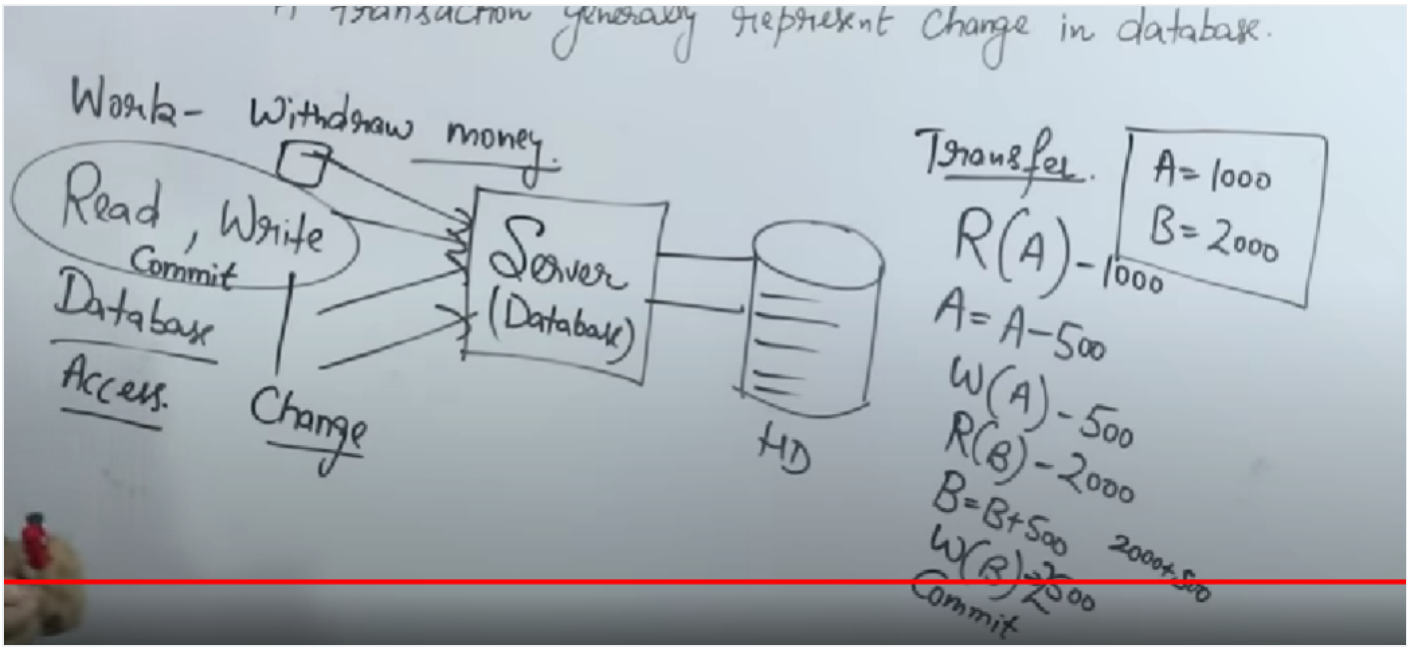


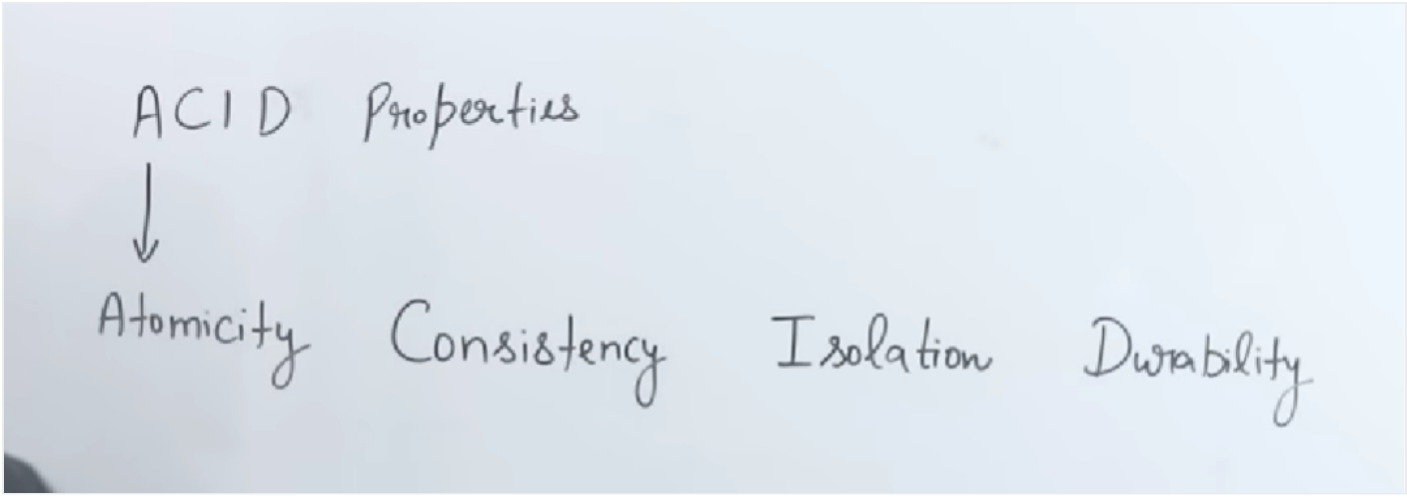
when we decompose the table to normalize it we also split the functional dependencies and if we take the union of all the functional dependencies then it should be equal to the original functional dependency.



Transaction management :







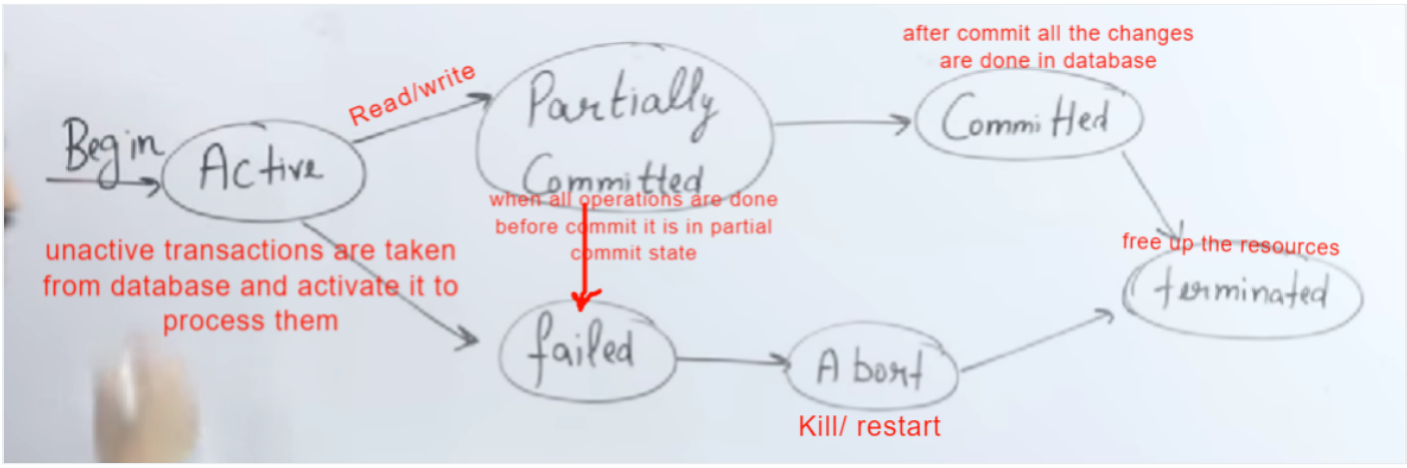
→atomicity (either all or none): if the transaction fails before the commit then we will roll back the transaction. A failed transaction can't be resumed it can only restart.

→consistency: before the transaction start and after the transaction completes, sum of the money or work should be same.

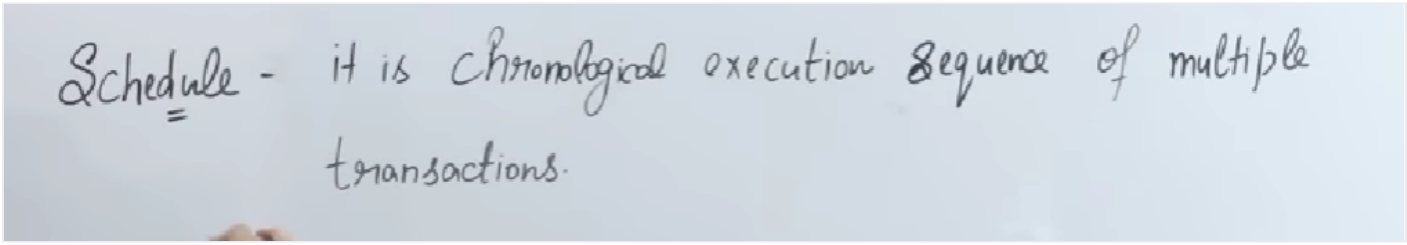
→isolation: to make a parallel transaction to a serial transaction, to make it consistent.

→durablity: all the changes in the database should be permanent.

Transaction states:



Schedule



schedule is, multiple transaction kis sequence mai execute ho rhi hai.

the schedule is of two types ⇒ 1. serial 2. parallel

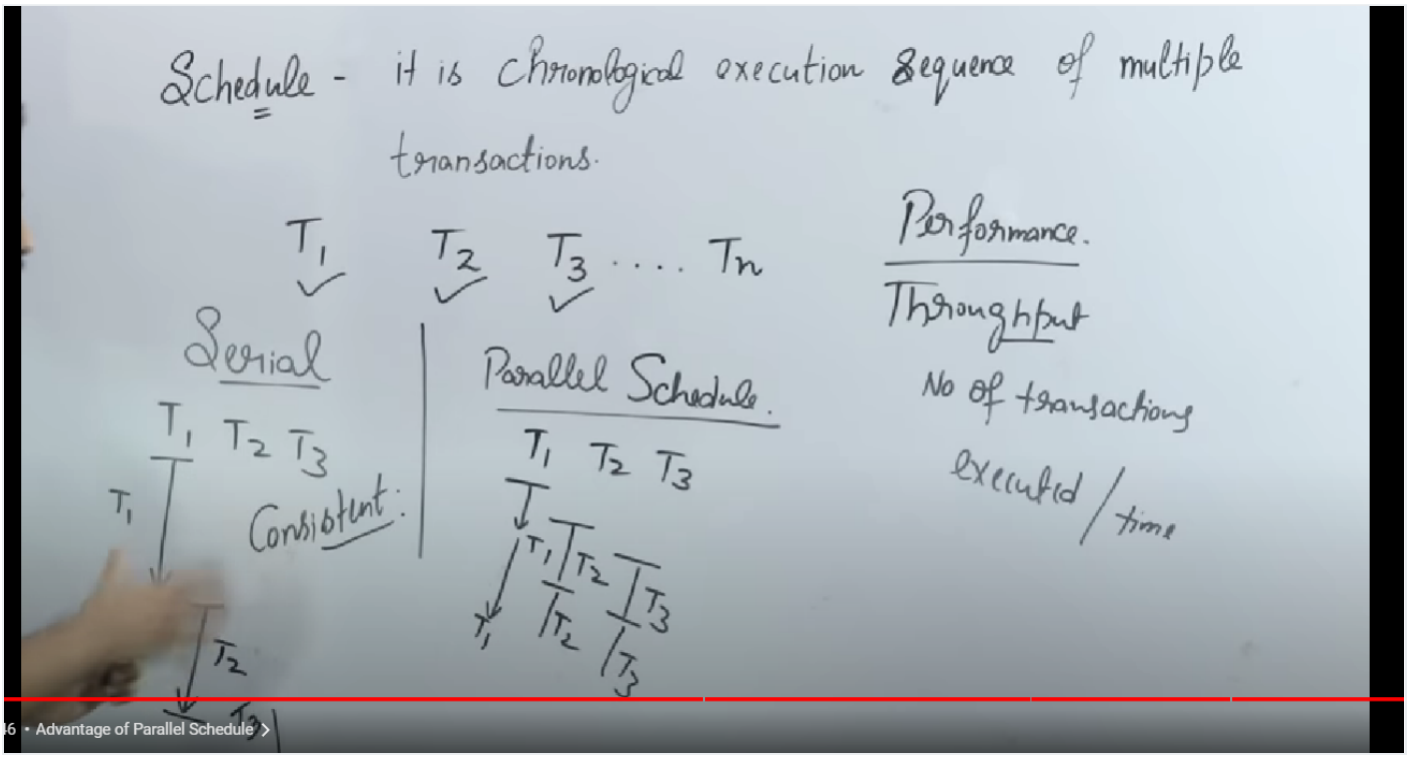
we categorize schedules into two things ⇒ serializability and recoverability.

⇒ serial: the execution of a transaction will be started only after the completion of previous transaction.

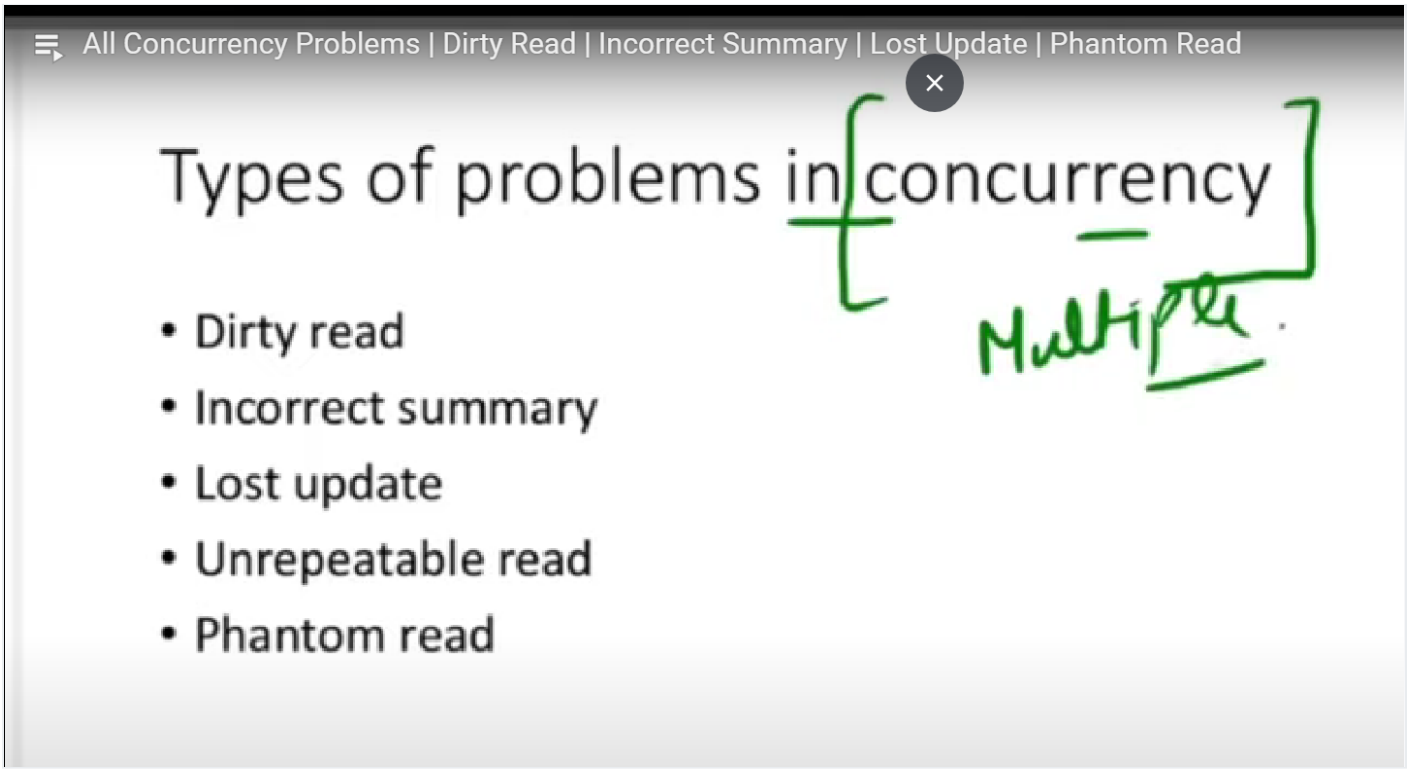
its advantage is consistency. the result is always consistent. and disadvantage is waiting time of transactions and performance is degraded.

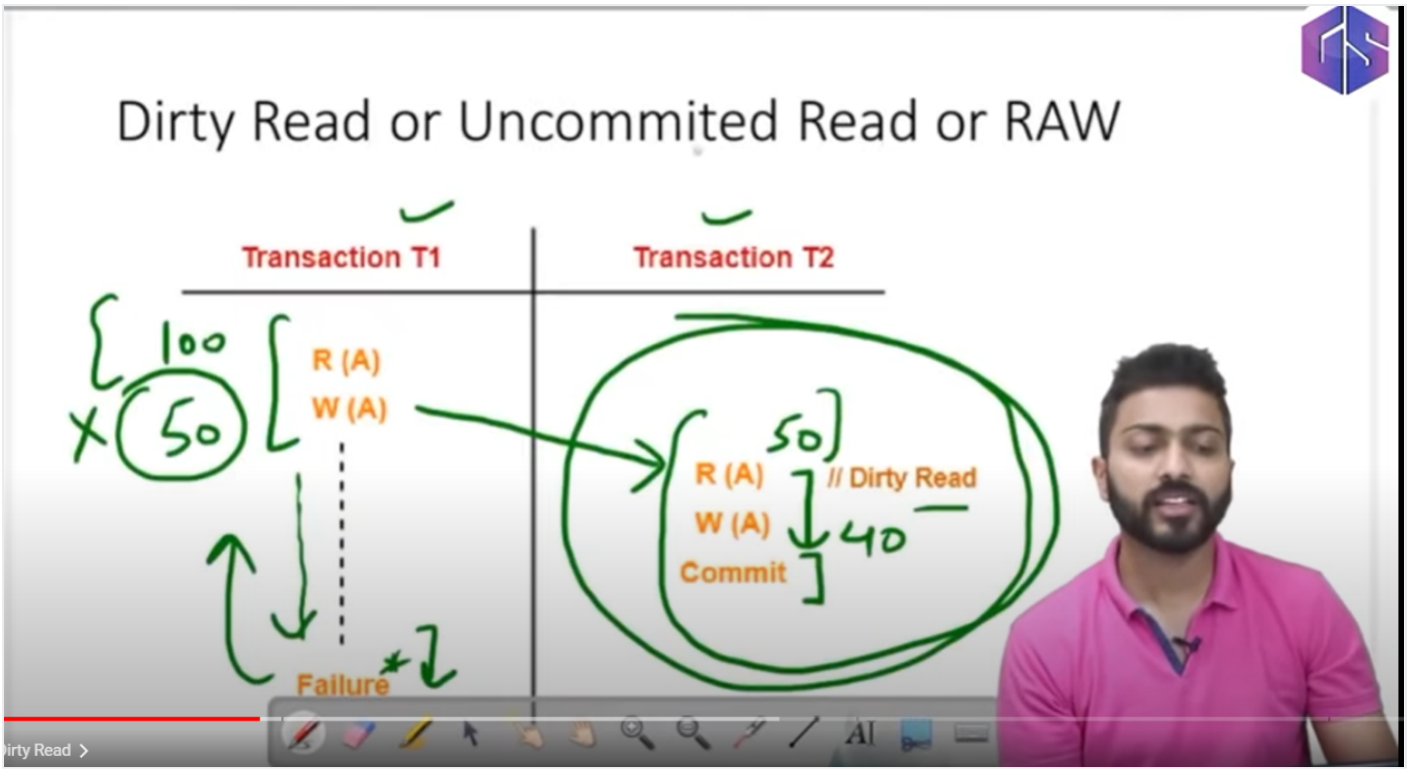
⇒ parallel: in this type of scheduling multiple transactions can come and start.

multiple transactions are executed at a single time. advanage : performance and throughput increases. disadvantage: inconsistency.

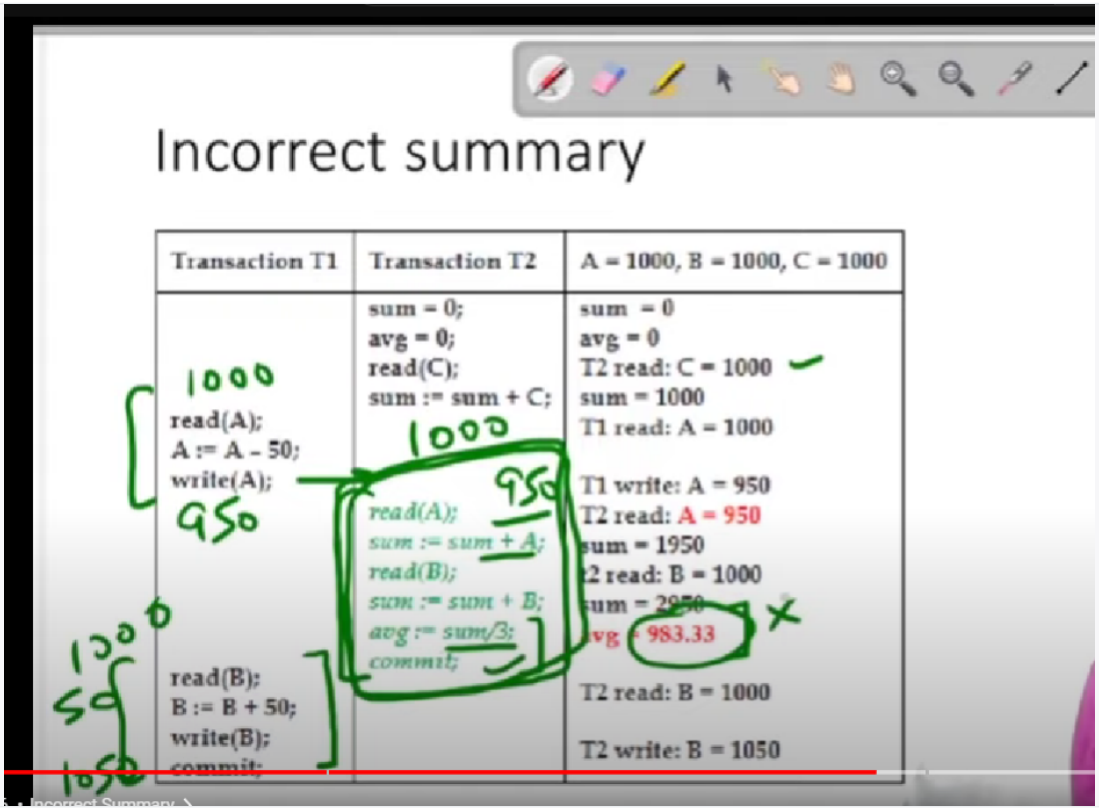


Concurency : multiple transactions in a single time (Parallel scheduling).

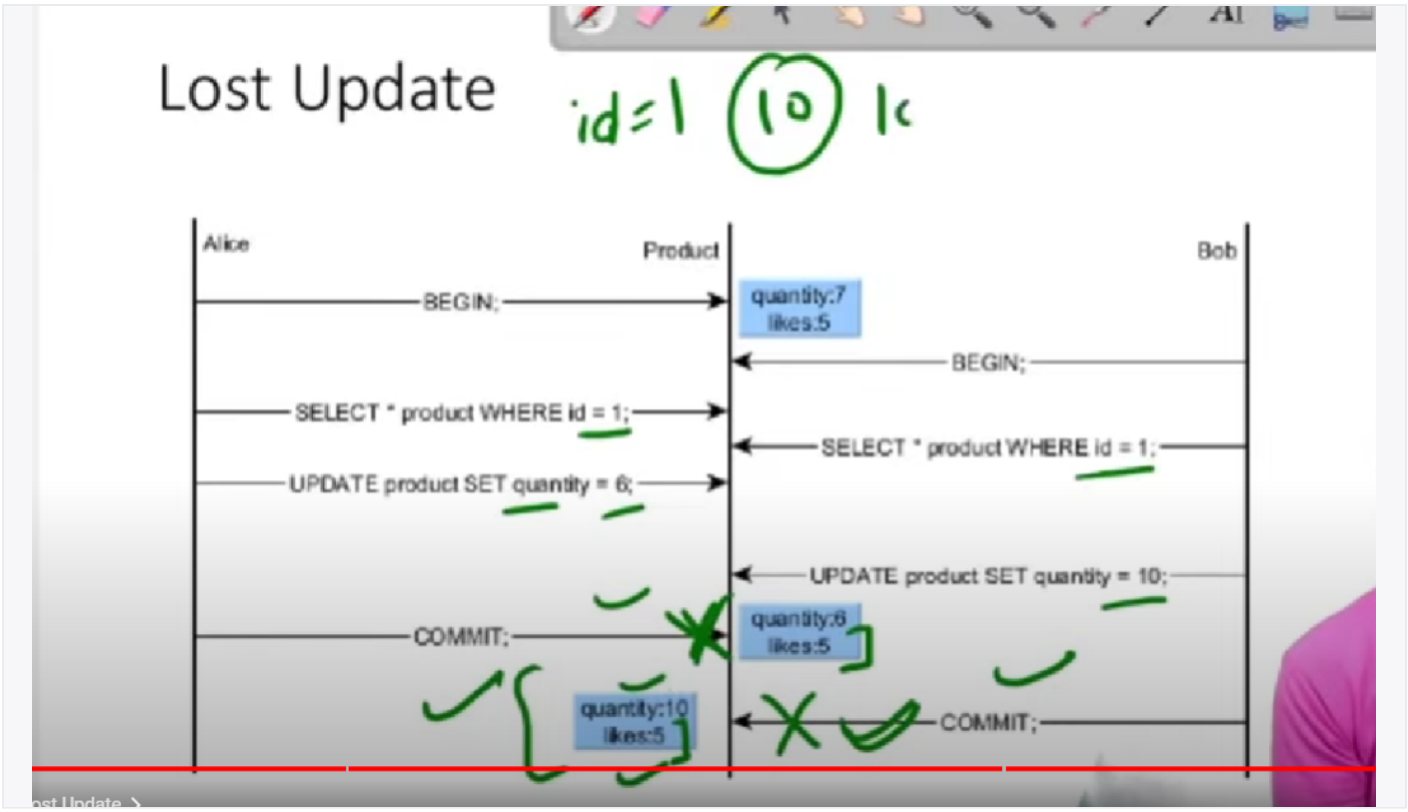




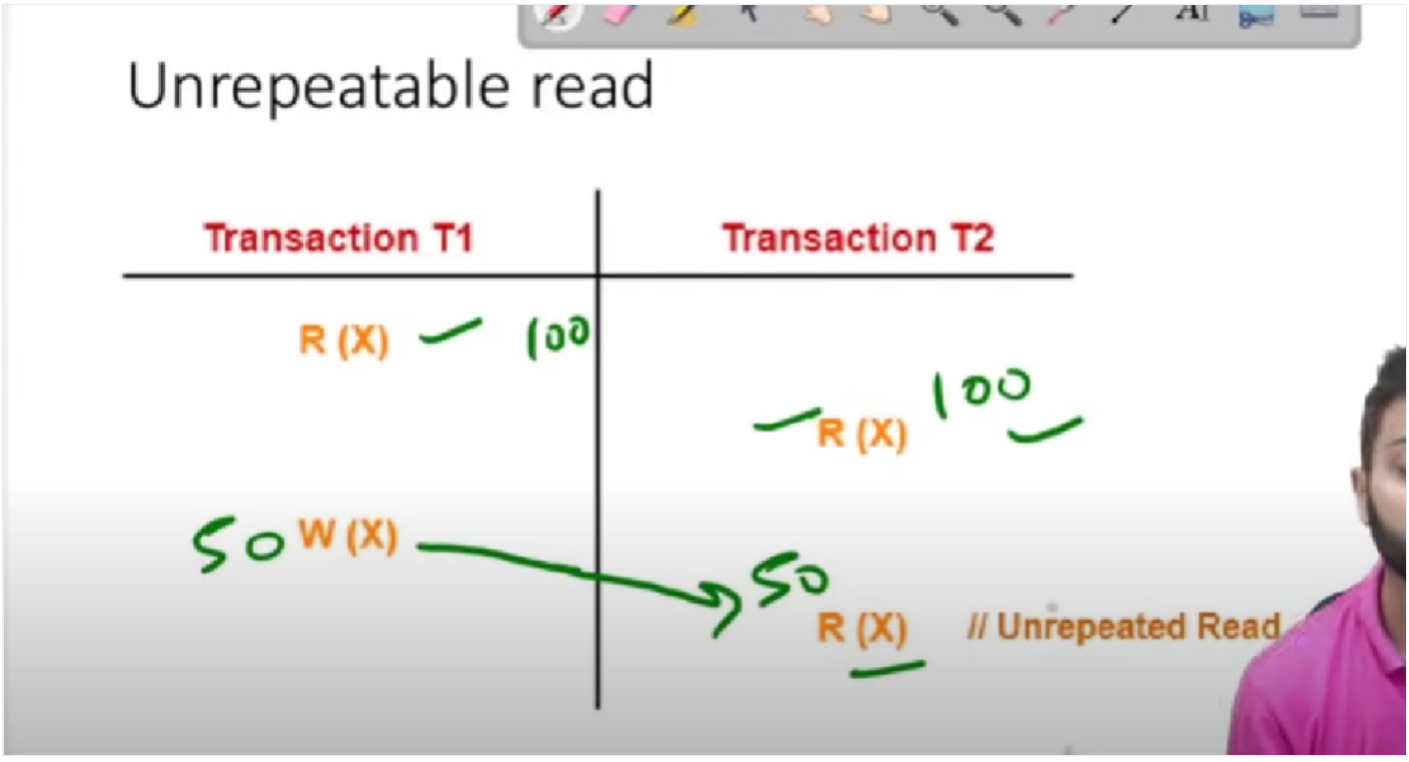
dirty read: a transaction t1 starts and did some changes and then t2 get it and did some changes and afterwards if t1 fails then it is roll back and t2 comes with dirty read as it make changes to a wrong value or variable.

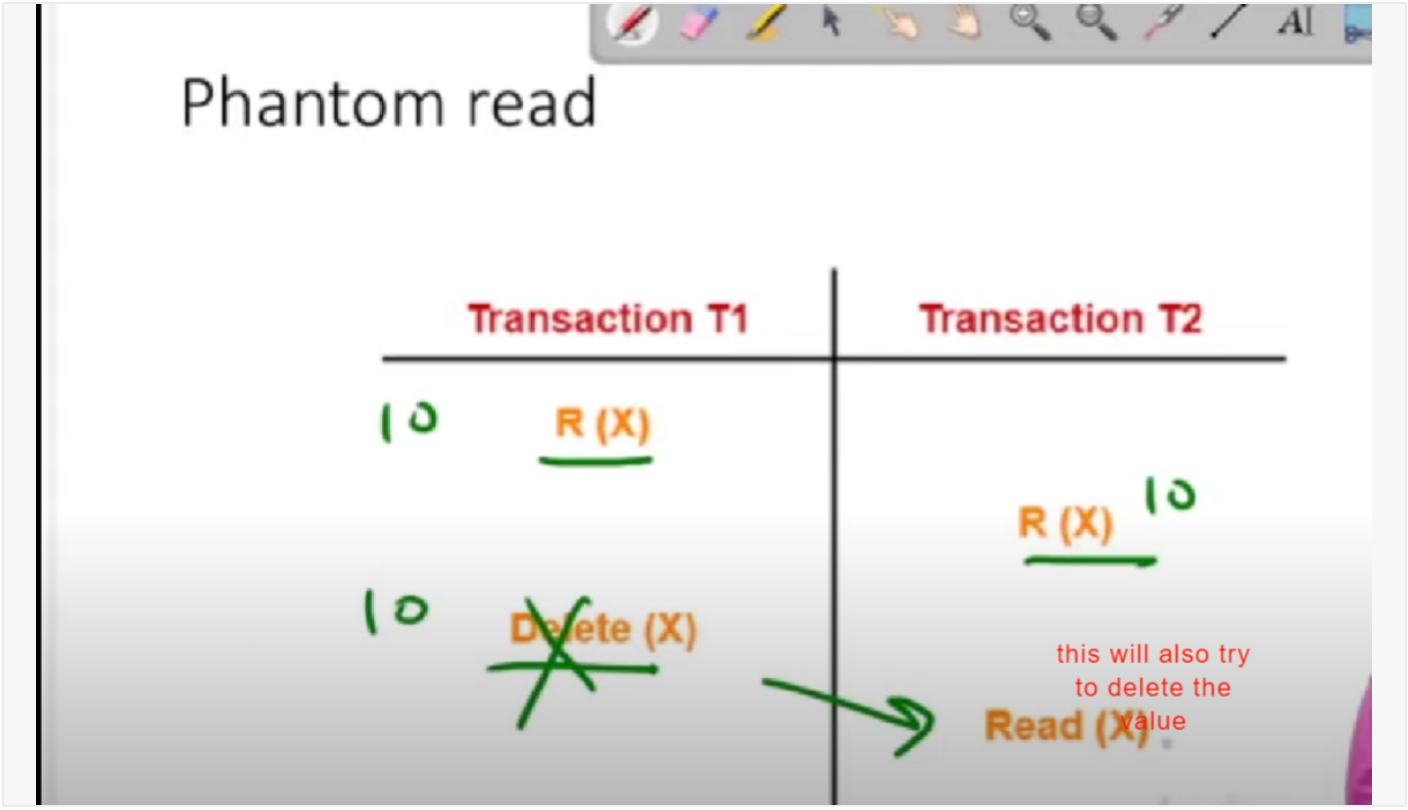


incorrect summary: this problem generally appears when a transaction was executing and then another transaction comes and claculated something in between which was waste or wrong.



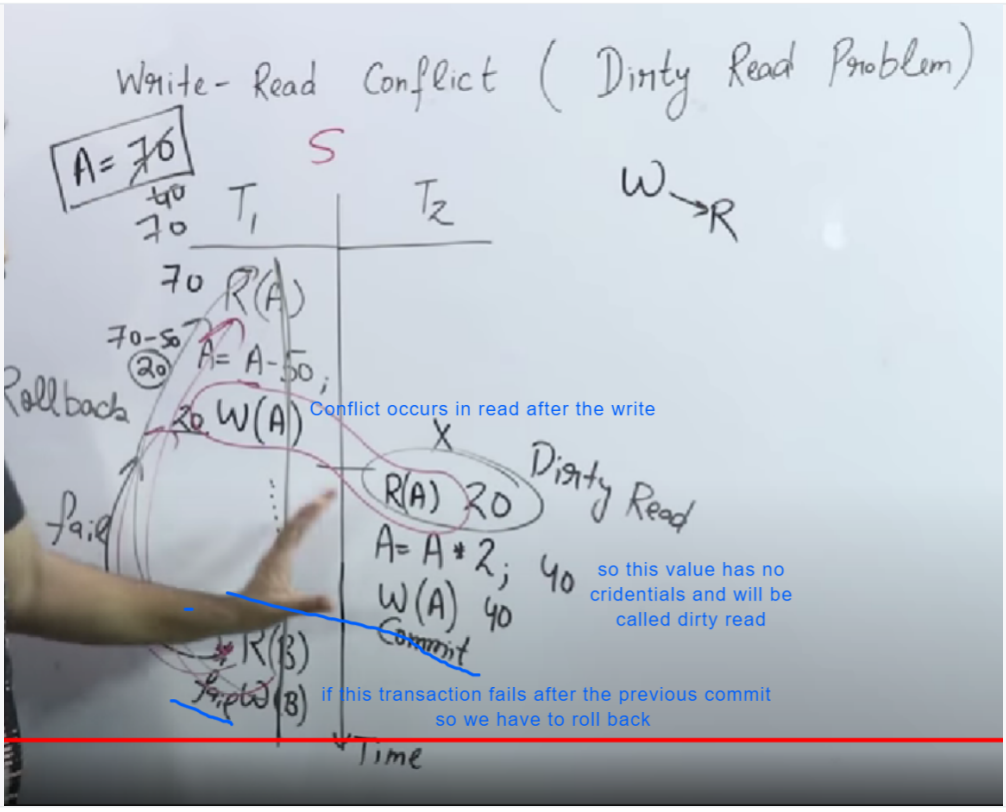
lost update: generally arises when someone updates the already committed transaction.



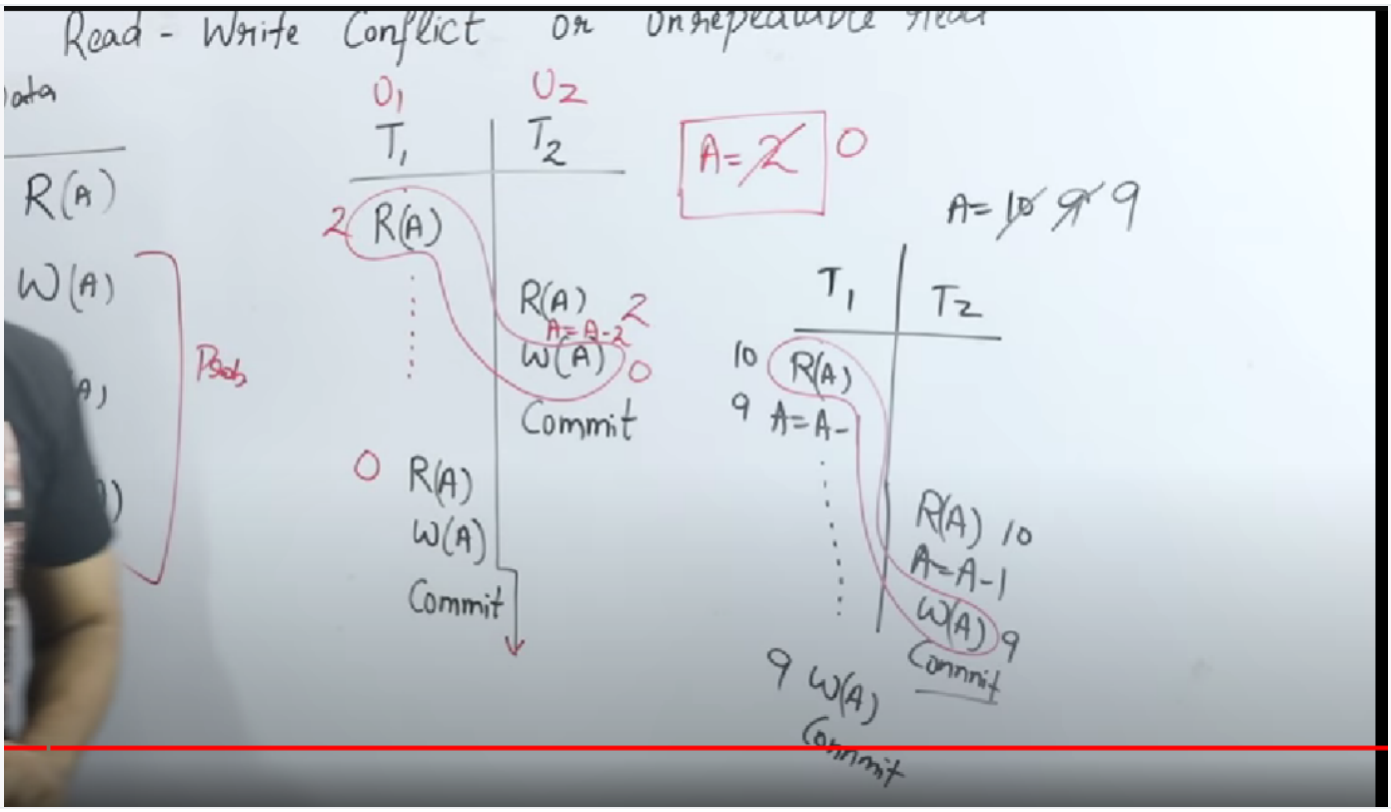


Write - read conflict

it occurs in parallel ransactions.

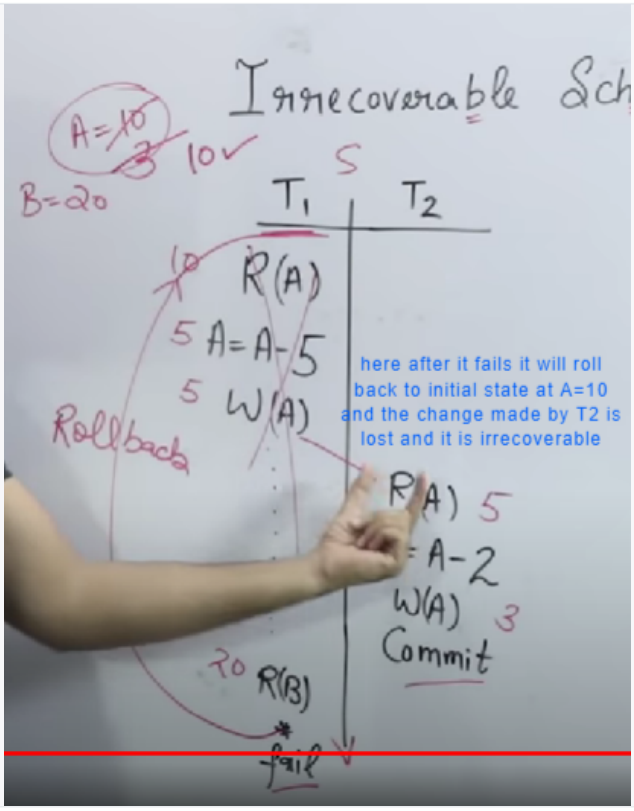


read-write conflict :



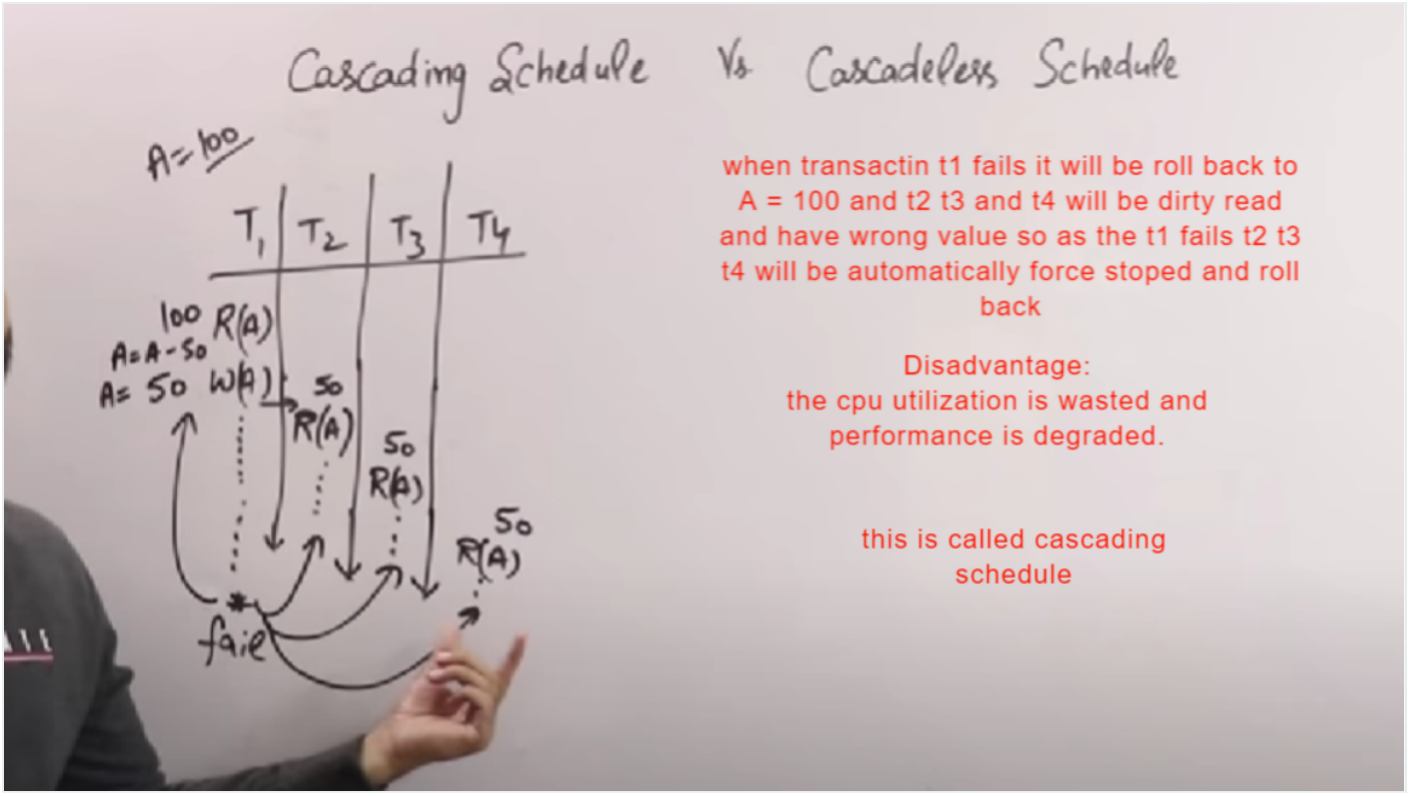
Recoverablity

irrecoverable schedule: a schedule in which a value can not be recovered.

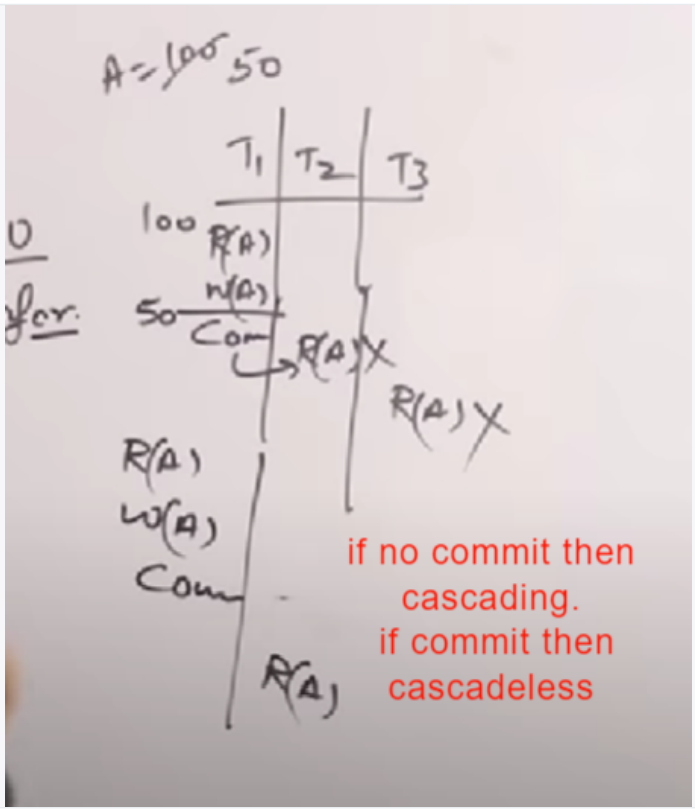


Cascading vs Cascadeless schedule

cascade: due to occurrence of one event multiple events are automatically occuring.



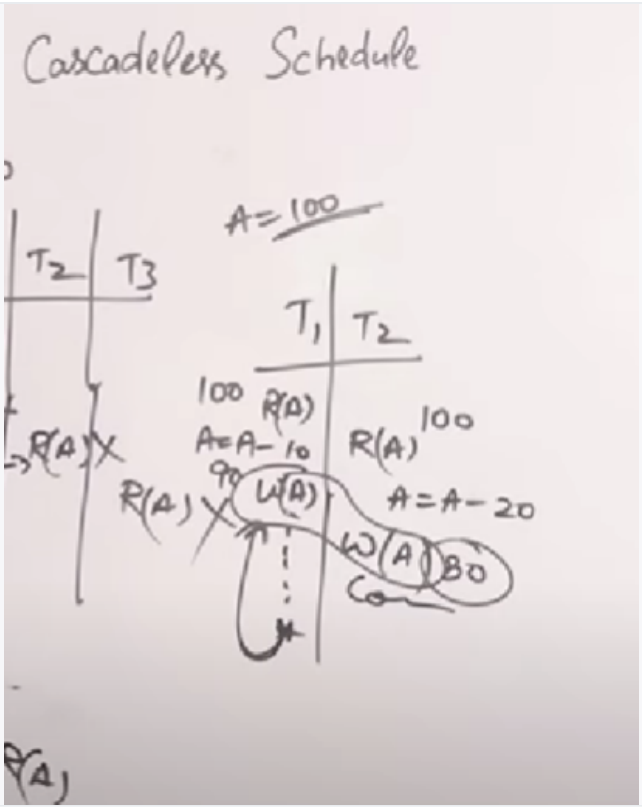
for cascadeless schedule: we will not allow the transactions to read from other transactions untill it is not committed.



A transaction can not **read** before commit but can **write** before commit.

there cannot be read after write on the same data until it is not committed.

but write after write can also occur problem as if t1 gets fail and it will roll back and the write done by t2 will be lost. it will be cascadeless but its can be lost update if t1 fails



so these problem is solved by strict cascading schedule.

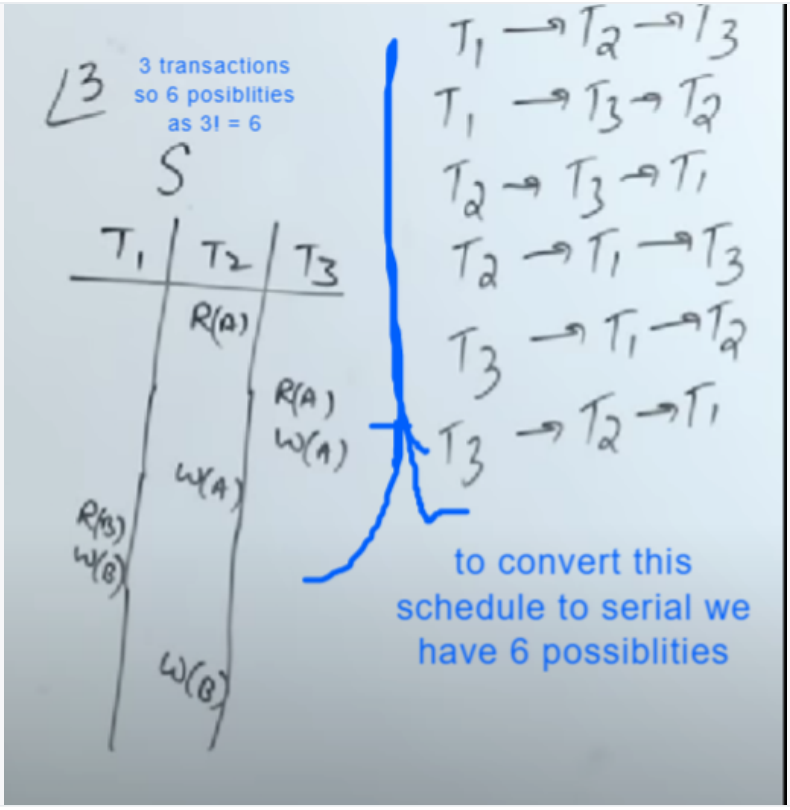
Serializablity

inter leaving of process is not done.

serial schedule: in this scheduling, the next process will start only after the completion of the first.

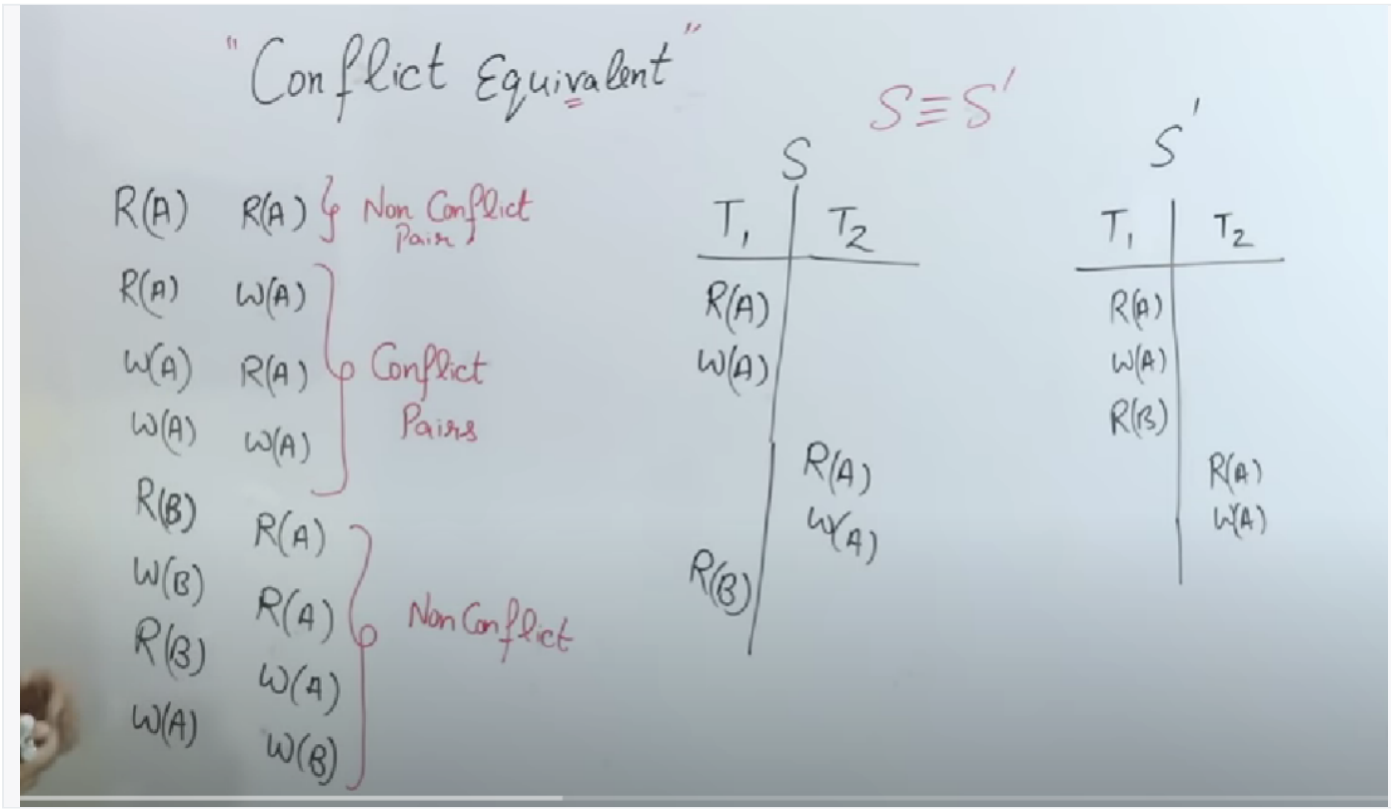
the serial transaction cannot be serialized only parallel transactions cannot be serialized.

we will be given a parallel schedule and we have to check the serializability that there exists a clone of that schedule that is in serial form.

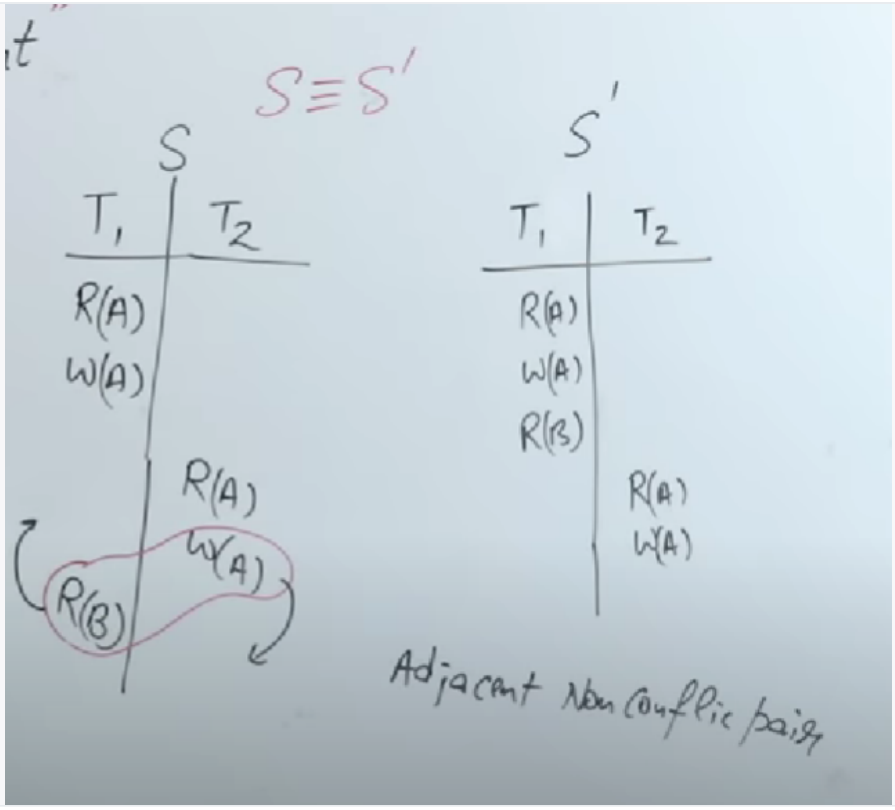


Serialization can be done in 2 ways: conflict equivalent and view equivalent .

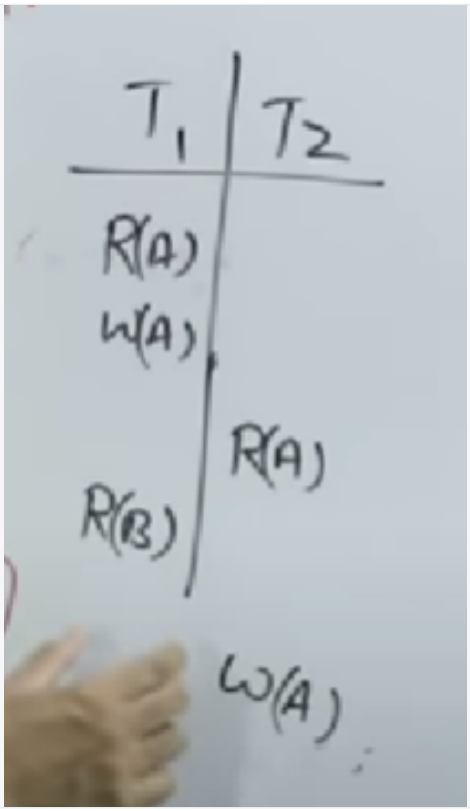
to check whether s and s' are conflict equivalent or not, we will check the pairs in below example:



we will check the adjacent pairs, if they are non-conflicting we can change their posititons.



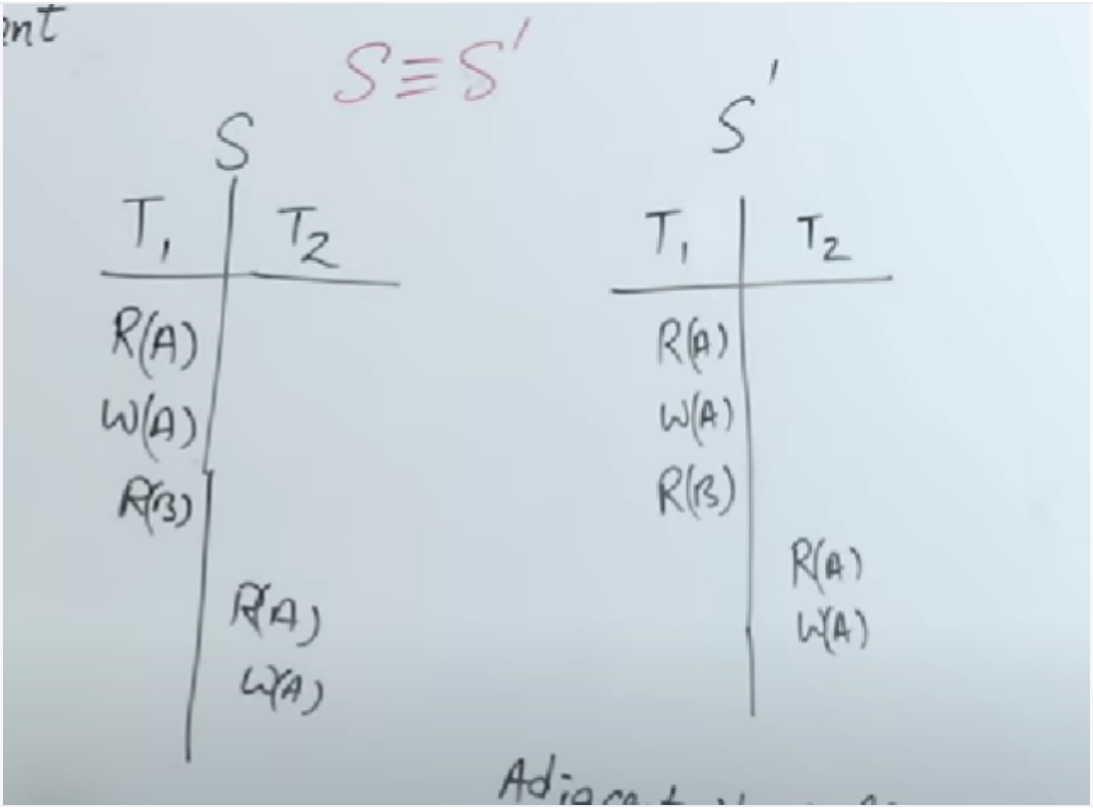
after changing the position the schedule will look like :



then again we will check the adjacent pairs.

as R(A) AND R(B) are non conflicting we will swap their positions.

now the schedule become

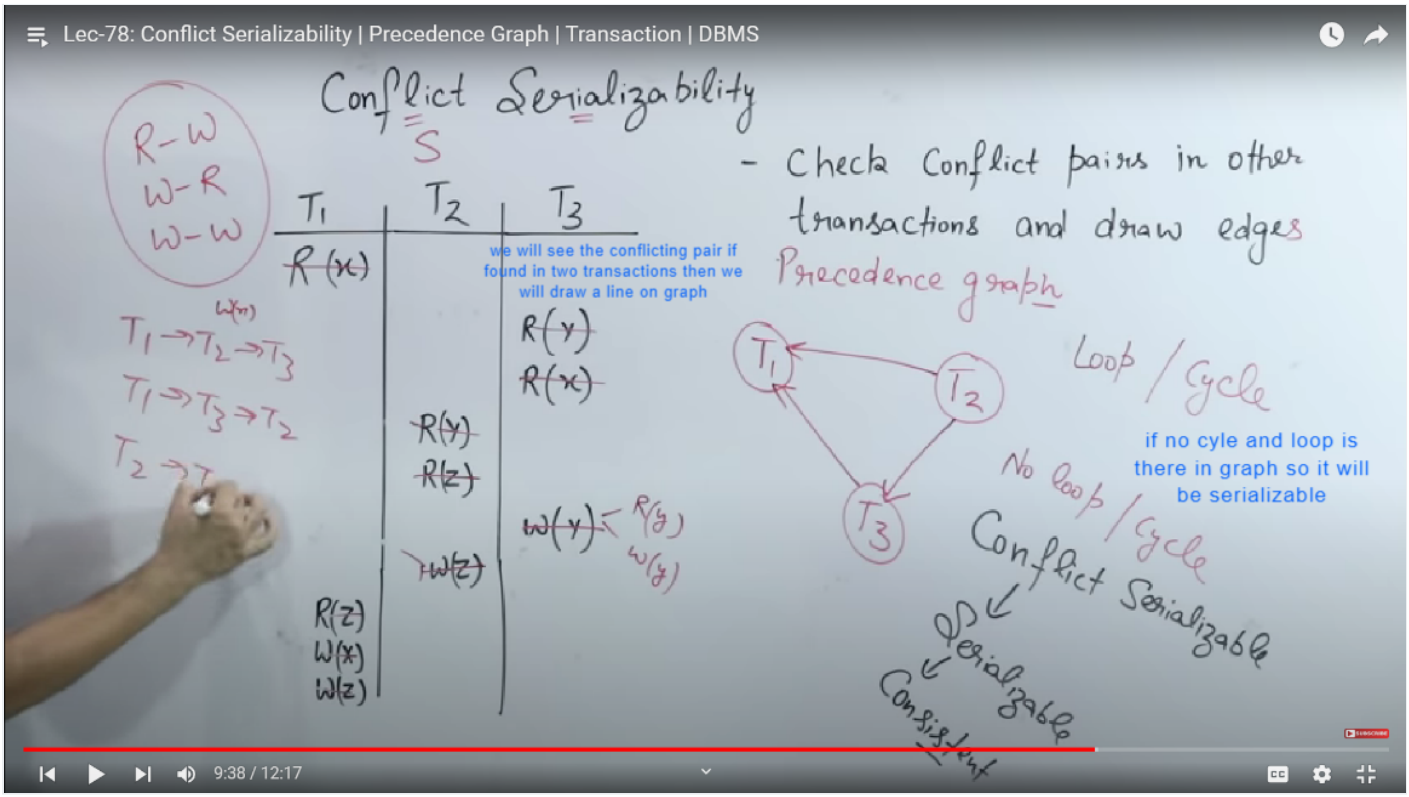


so s and s' are conflict equivalent schedules.

⇒ if there is a case of conflicting pairs like R(A) and W(A), there will be no swapping and no change in position.

⇒ for any schedule whose conflict equivalent exists, it is serailizable.

Q.

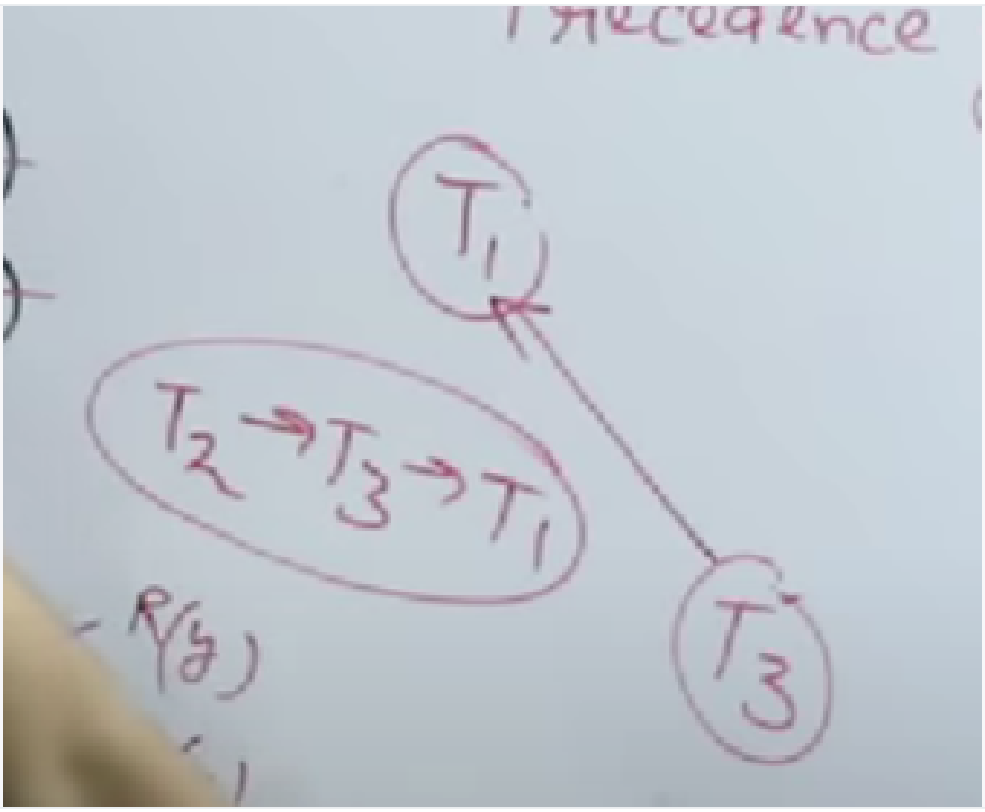


to know the sequence of the serial schedule we will take the graph and take the transaction with minimum indegree (number of arrows coming inward, starting with 0) and it will be the sequence. as in above example:

T2 as indegree 0, it will be first and taken out of graph

now there will be two transaction left.

now,



T3 has indegree = 0, it will be second

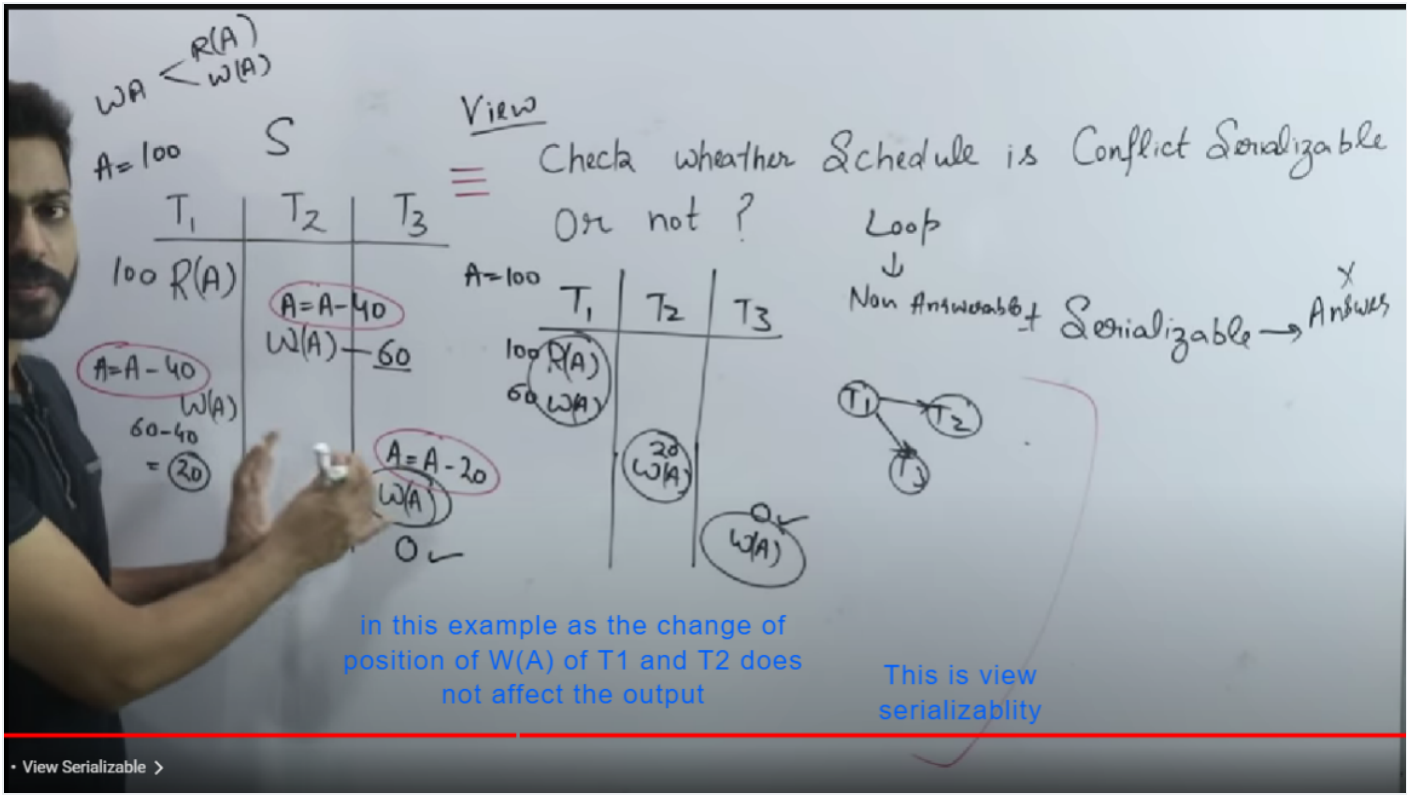
and last is T1.

so the sequence will be T2→T3→T1.

⇒ if in the graph there is a loop, then it is not **conflict serializable.**

but we cannot say it is not serializable, so we use view serializable.

Q.

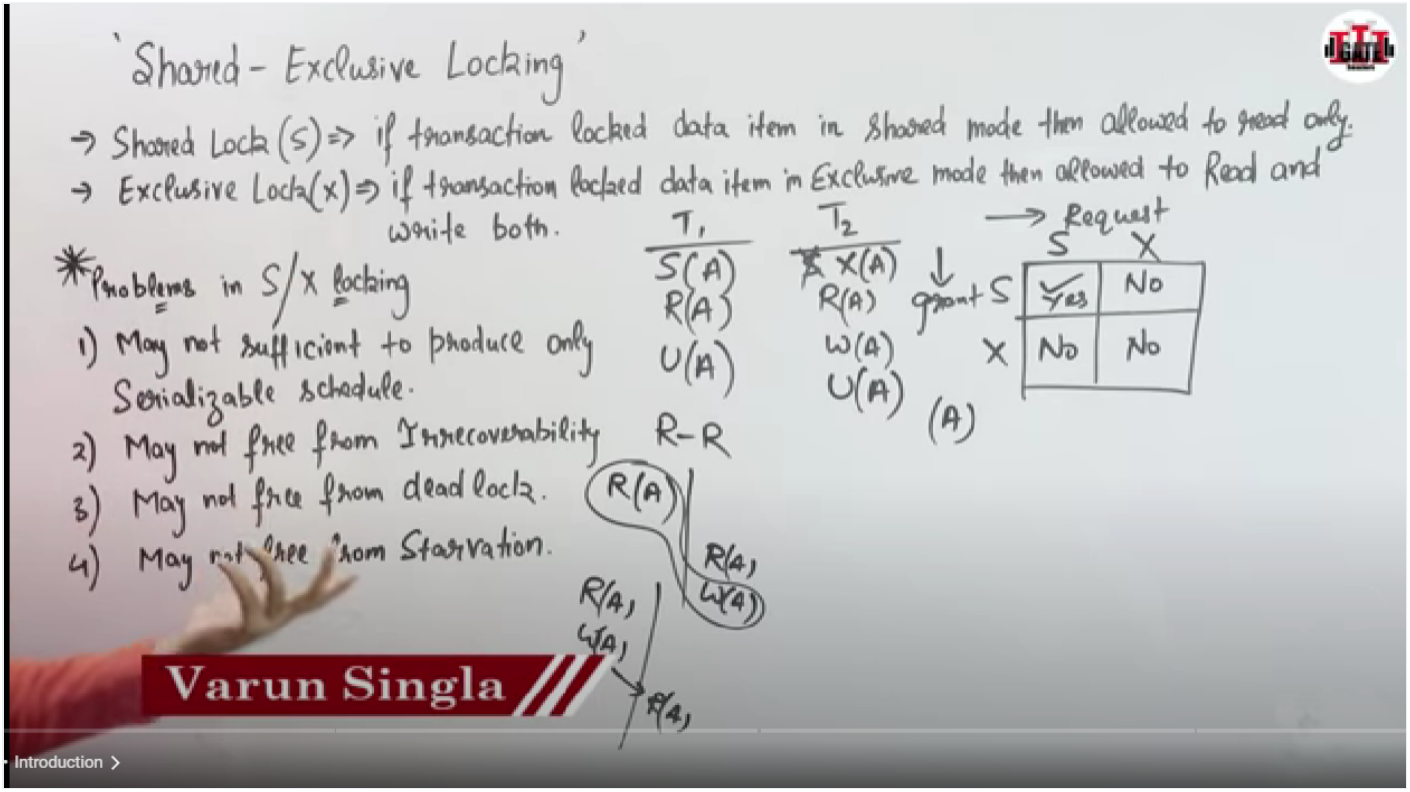


if the output operation states is fixed we can interchange the position of in between operation in view serialization.

Concurrency control protocols

the basic aim is to achive serializability and recoverability and make concurent (parallel) schedules serial.

achive by using **locking protocol.**



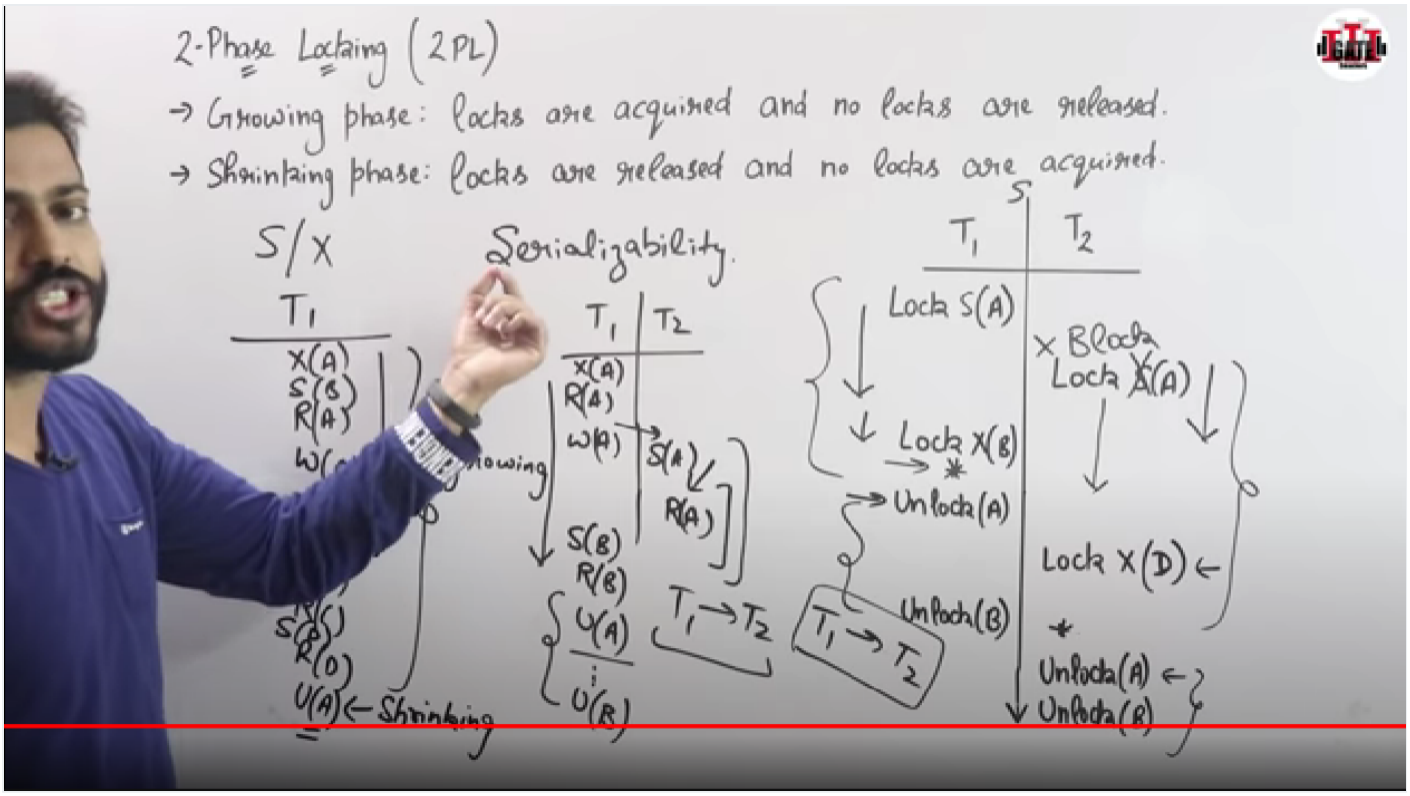
2 phase locking

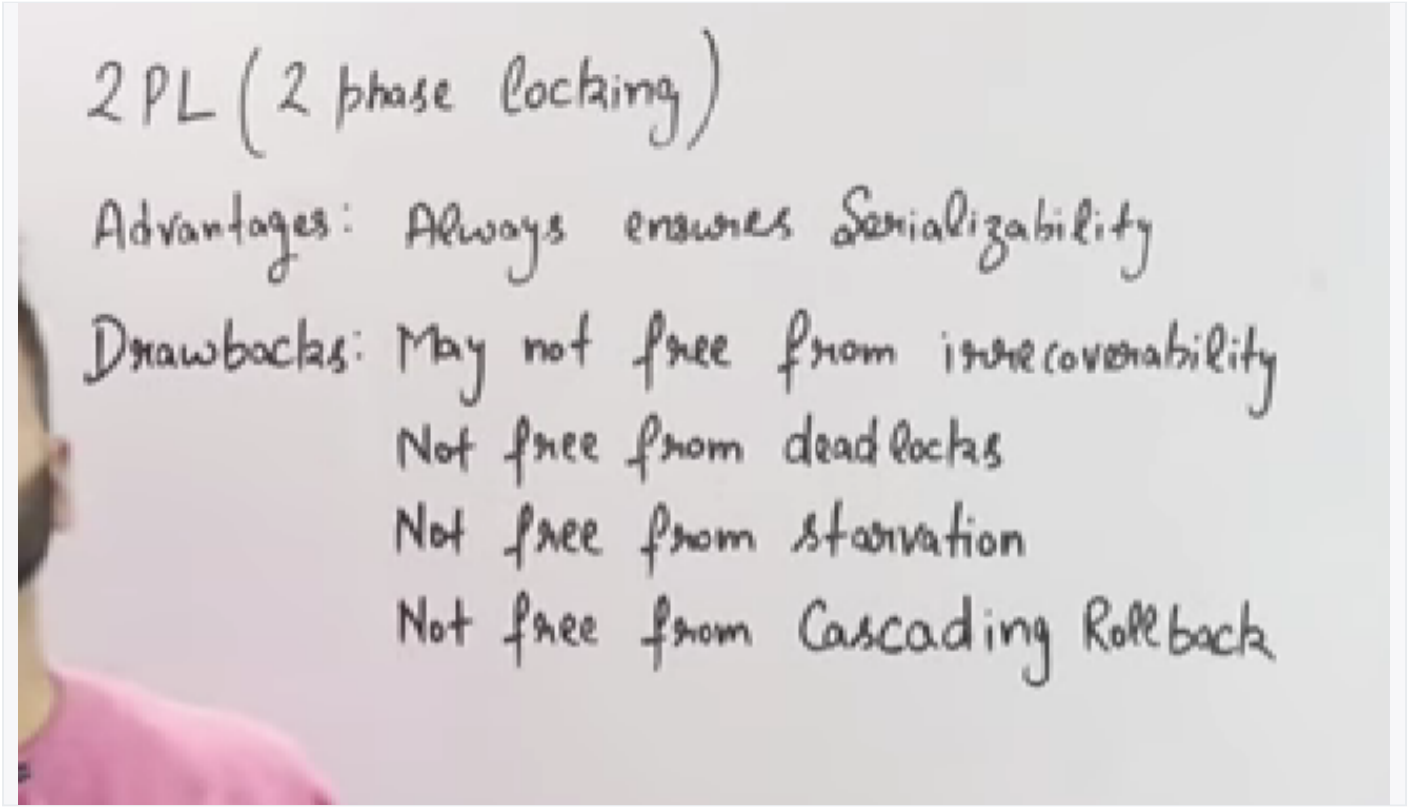
use to achive serializablity.

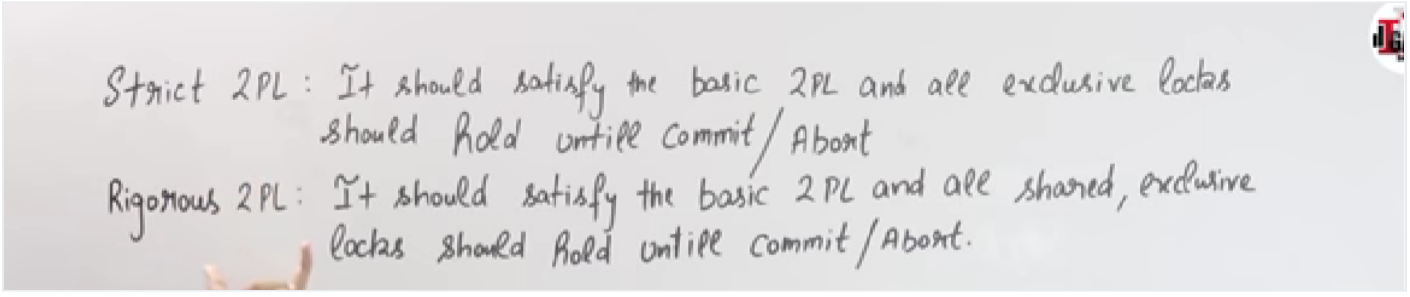
the transaction who is following 2PL is always serailizable.

sequence can be given by lock point.

the point just before the first release of the lock in transaction. whose lock point comes first will start the sequence and the sequel will go on.







these two are recoverable and free from cascading rollback(will be always cascadeless).

conservative 2PL : in this a transaction will take all the locks for the operations. so it is free from starvation and deadlock.

it is not practical in real life.