18/9/18:

Q

A	B	C
1	1	3
2	1	3
2	3	1
1	2	1
1	4	2
4	2	1

Find the non-trivial dependencies:

A → A ✓	B → A X	C → A X
A → B X	B → B ✓	C → B X
A → C X	B → C ✓	C → C ✓
A → BC X	B → AC ✓	C → AB X

AB → C ✓
AC → B ✓
BC → A X

Process of Normalization to reduce anomaly:

1NF 2NF 3NF BCNF 4NF 5NF

NF → Normal form.

4NF, 5NF → doesn't exist for real-time database

Attribute domain shouldn't change and unique name for Attributes/Columns.

1NF:-

In this no attribute should be a multi-valued attribute. and the order in which data is stored does not matter.

Eg:-

Sid	Sname	Cname
1	A	C/C++
2	B	C/Java
3	C	Java.

We can write it as:-

sid	Sname	Cname
1	A	C
1	A	C++
2	B	C
2	B	Java
3	C	Java.

So now our table satisfies the first Normal form.

→ Using the first NF, data redundancy increases, as there will be many columns with same data in multiple rows but each row as a whole will be unique.

Sid → Sname
Sid → Cname (X)
Sid → Cname Sname (X)
Sid Cname → Sname (✓)

If we divide the table in 2 or more table to remove redundancy it is known as decomposition.

If want, can skip.

* Extra:

Eg:-

Rollno	name	brach	HOD	Office tel
1	AB	CS	X	12
2	BB	CS	X	12
3	CB	CS	X	12
4	DB	CS	X	12

In the above table, we have data of 4 CS students, There is data redundancy as Branch, HOD & office tel are repeated for students in same branch.

① Insertion Anomaly

① Suppose for a new admission, until and unless a student opts for a branch, data of the student cannot be inserted, or else we will have to set the branch information as NULL

② Also, if we have to insert data of 100 students of same branch, then the branch information will be repeated for all those 100 students.

These scenarios are insertion anomalies

② Updation anomaly:

What if X leaves the college? or is no longer the HOD of CS? In that case all the student records will have to be updated, and if by mistake we miss any record, it will lead to data inconsistency. This is updation anomaly.

③ Deletion Anomaly:-

In our student table, two different info are kept together, Student info and Branch info. Hence, at the end of the academic year, if student records are deleted, we will also lose the branch info. This is deletion anomaly.

Normalization of database:

It is a technique of organizing the data in the database.

It is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like insertion, Update & delete anomalies.

It is used for mainly two purpose
① Eliminating redundant (useless) data
② Ensuring data dependencies.

② 2NF

A relation is in 2NF if
① It is in 1NF.
② There should be no partial dependency.

[Can skip]

Dependency:- Suppose I have a student database, then all I need is Sid and every other column depends on it, or can be fetched using it.
This is dependency or we call call it functional dependency.

→ Consider previous student table.
→ Now let's create another table subject.

Sub-id	Sub-name
1	Java
2	C++
3	Php.

→ Let's create another table Score.

score-id	Stud-id	sub-id	marks	teacher
1	10	1	70	Java teacher
2	10	2	75	C++ "
3	11	1	80	Java "

In this table Stud-id + sub-id together forms a candidate key (Hence PK).

In the above table, we have a column name teacher which is only dependent on subject.
This is partial dependency, where an attribute in a table depends on only a part of the primary key and not on the whole key.

The simplest solution is removing teacher's name from score table and add to Subject table.
So, now our score table is in the second normal form, with no partial dependency

In short:

① PD exists, when for a composite PK, any attribute in the table depends only on a

part of the primary key and not on the complete primary key.

② To remove partial dependency, we can divide the table, remove the attribute which is causing partial dependency, and move it to some other table where it fits in well.

Class notes:

If R(X, Y, A)
- X → CK
- Y → subset of X
- A → non-prime attribute

If X → A then it is full functional dependency.

and Y → A exists then it is partial functional dependency

R(A, B, C, D) → If A, B are super key & candidate key then they are prime attributes & CD are non prime attribute.

Sid → Sname (partial).

Sid, cname → Sname (Full).

R(A, B, C, D, E, F)

BC, A → Key

A → BCDEF

BC → ADEF

B → F

D → E

Here B is part of key and F is non prime attribute. This violates 2NF.

D → E

Both are non-prime attribute, so here we cannot apply the test, so it is by default 2NF.

Making it 2NF:

R_1		
A	BC	DE

A → BCDE

BC → ADE

D → E

R_2	
B	F

B → F

Decomposition says production that violates should be given another table.

3NF → To satisfy 3rd normal form either LHS of an FD should be super key or RHS should be prime attribute

[can skip]

1. It should be 2nd NF.
2. And it should not have transitive dependency.

Now suppose to our score table if we add exam_name & total_marks

Our new column exam_name depends on both student and subject.
The column total_marks depends on exam_name as with exam type the total score changes.
But, exam_name is just another column in the score table. It is not a primary key or even a part of the primary key, and total_marks depends on it.

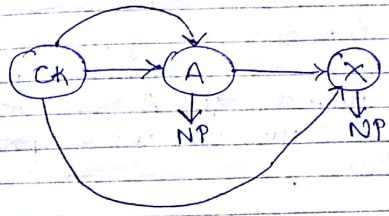
This is transitive dependency: When a non-prime attribute depends on other non-prime attributes rather than depending upon the prime attributes or primary key.

Solution:- Take out the columns exam_name & total_marks from Score table & put them in an Exam table & use the exam_id wherever required.

Advantage of removing Transitive dependency

1. Amount of data duplication is reduced
2. Data integrity achieved.

Class notes



i.e, if CK → A and A → X } should not be there.

In the previous relation, we found out that NP → NP should not exist as in that transitivity is true

i.e BC → A
BC → D
BC → E
& D → E

Now we will make another table for D → E.

R_1 a	R_1 b	R_2
A BC D	DE	B F
A → BCD	D → E	B → F
BC → AD		

Boyce - Codd Normal Form (BCNF)

Def^n

① It should be in 3rd N.F.

② And, for any dependency A → B, A should be a super key.

(In simple words, it means that for a dependency A → B, A cannot be a non-prime attribute, if B is a prime attribute.

Eg:-

Stud id	Subject	Professor
101	Java	P. Java
101	C++	P. Cpp
102	Java	P. Java2
103	C#	P. Chash
104	Java	P. Java

In the table above :-

- One student can enrol for multiple subjects.
- For each subject, a prof. is assigned to the student.
- There can be multiple prof. teaching one subject.

In the above table, student_id, subject together form the PK.

Here, One prof. teaches only 1 subject, but a sub can have 2 prof.

Here subject depends on prof. name.

This table satisfies all the three normal forms but not BCNF.

Why ??

Here subject is a prime attribute as Student-id, subject form primary key.

There is one more dependency

Professor → subject.

while subject is a prime attribute, prof. is a non-prime attribute, which is not allowed by BCNF.

To make this BCNF we will decompose this table into 2 tables.

Student table:

Sid	p-id
101	1
101	2
and soon.	

Professor table:

p-id	Professor	Subject
1	P. Java	Java
2	P. Cpp	Ctt
and soon..		

Now it satisfies BCNF.

Eg of BCNF in terms of relation.

R(A, B, C, D)

A → BCD.
BC → AD.
D → B.

3rd Normal Form.
keys A, BC.

A → super key. (In 1st dependency)
BC → super key (In 2nd ")

but in, D → B D is not a key, but B is a prime attribute.

Hence we can break our relationship R into two relationships R_1 and R_2

R(A, B, C, D)

R_1(A, D, C) R_2(B, D)

Class note:

X → A where X is a Super key

Statement:

* Every relation R which is in BCNF is also in 3NF (True).

* Every relation which is in 3NF is also in BCNF (Not necessarily).

Q R(S, C, I)
SC → I
I → C
Find the highest normal form.

$[SC]^+ = \{I, S, C\}$.

$[I]^+ = \{I, C\}$.

SC is super key

$SC \rightarrow I$ (BCNF)

$I \rightarrow C$ (3NF).

C ↘ prime

I ↙ non prime

Hence, highest normal form is 3NF.

19/9/18

ABC Keys = AB, BC.

$AB \rightarrow C$ BCNF

$C \rightarrow B$ 3NF.

Eg.

A	B	C
1	X	Y
2	Xa	Y

Then C can't identify B because there are two values corresponding to Y.

Decomposition Properties:

① Attribute preservation.
When we decompose all tables, all the attributes in the original table should come in the decomposed table.

② Lossless decomposition.

R → Original table

$R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n = R$,

where $R_1, R_2 \cdots R_n$ are decomposed tables.

There should be no generation of false tupples.
(Tupples should be same as original)

③ ~~Dependency~~.

Eg. R

A	B	C
1	1	2
2	1	3
3	2	1

R_1 (AB)

A	B
1	1
2	1
3	2

R_2 (BC)

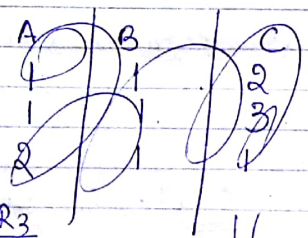
B	C
1	2
1	3
2	1

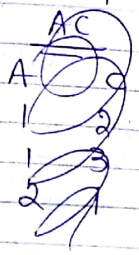
$R_1 \bowtie R_2$ - (AB, BC)

A	B	C
1	1	2
2	1	2
1	1	3
2	1	3
3	2	1

} False tuples Because join over non-primary key

(AB) ⋈ (AC)





R_3

A	B
1	1
2	1
3	2

R_4

A	C
1	2
2	3
3	1

$R_3 \bowtie R_4$

A	B	C
1	1	2
2	1	3
3	2	1

No. ~~tuples~~ false tuples as join over primary key A.

Lossless properties:

R(ABCD)

R_1(AB) R_2(CD)

① All attributes should be preserved

② $R_1 \cap R_2 \neq \phi$

③ $R_1 \cup R_2 \rightarrow R_1 - R_2$

or

$R_1 \cup R_2 \rightarrow R_2 - R_1$

Q R(ABCD)

R_1(AB) R_2(BCD)

A → B
B → C
C → D.

$R_1 \cap R_2 \neq \phi$ (as B is common).

$[B]^+ = \{B\ C\ D\}$.

As $[B^+]$ = can be the key of R_2 table so we can join them

R_{12}(ABCD) → lossless.

(we can join R_1 & R_2 and R_{12} is lossless)

→ Main condition is to check if the common attribute b/w 2 tables can be the key for one decomposed table, then we can join these tables and it is lossless.

Q
$AB \rightarrow C$
$C \rightarrow A$
$C \rightarrow D$

$R(ABCD)$
$R_1(AB)$ $R_2(ACD)$

$[AB]^+ = \{A, B, C, D\}$.

$[C]^+ = \{A, C, D\}$

$[A]^+ = \{A\}$.

So it is lossy, as common attribute doesn't have all the attributes of any table

Q
$R(ABCDE)$

$R_1(ABC)$ $R_2(ACDE)$

$A \rightarrow BC$
$CD \rightarrow E$
$B \rightarrow D$
$E \rightarrow A$

$[AC]^+ = \{A, B, C, D, E\}$.

So it is not lossy as it can be the key of both table (Our condition is should be key for any table and here it can be key for both table). So it is not lossy i.e., lossless property.

Q
$R(ABCDEG)$

$R_1(AB)$ $R_2(BC)$ $R_3(ABDE)$ $R_4(EG)$.

~~If we combine R.~~

$AB \rightarrow C$
$AC \rightarrow B$
$AD \rightarrow E$
$B \rightarrow D$
$BC \rightarrow A$
$E \rightarrow G$.

When we combine R_1 & R_2
$[B^+]^+ \rightarrow \{B, D\}$ (Doesn't cover any table)

So it is lossy.
We will not consider R_1 & R_2.

Combining R_1 and R_3.

$[AB]^+ = \{A\ B\ C\ D\ E\ G\}$.

So we can combine R_1 & R_3.

$R_{13} = (A\ B\ D\ E)$.

Now if we combine R_{13} & R_4.

R_{13} & R_4.

$[E]^+ = \{E, G\}$ {so it can be the key for R_4}

So this is also lossless decomposition and we can combine R_{13} & R_4

$= R_{134}\ (A\ B\ D\ E\ G)$.

Now we are left with R_2.

Combine R_{134} & R_2.

Common attribute = B.

$[B]^+ = \{B, D\}$

This doesn't conver any of the two table - R_{134} & R_2 so this is lossy decomposition and we won't combine R_{134} & R_2.

Q $R_1(ABC)$, $R_2(ACDE)$, $R_3(ADG)$.

Same dependency as previous question

① Combining R_1 & R_2.

$[AC]^+ = \{A, C, B, D, E, G\}$.

So it covers both R_1 & R_2

So this is lossless decomposition and now we can combine R_1 & R_2

$R_1R_2 = (A\ B\ C\ D\ E)$.

② Combining R_1R_2 & R_3.

$[AD]^+ = \{A, D, E, G\}$.

So it covers R_3. It is also lossless decomposition. and we can combine R_{12} & R_3 R_{123} $(ABCDEG)$.

6+1

Q. Name, CNO → Grade. BCNF
Roll No, CNo → Grade BCNF
Name → Roll No. 3NF
Roll No → Name. 3NF

Highest Normal Form = 3NF.

Q. Book (T, A, CNo, P, Y, Price).
Collection (T, A, CNo).

TA → CNo (BCNF)
CNo → TAPY (BCNF)
PTY → Price. (3NF)

2NF because this is partial PK. CS PTY is neither superkey nor Price is prime attribute so it is in 2nd NF

[TA]+ = {T, A, CNo, P, Y, Price}
[CNo]+ = {CNo, T, A, P, Y, Price}.

Our key is CNo and [TA]+

[PTY]+ = {P, T, Y, Price} (X)

Highest normal form = 2NF.

☆ When extra attribute are there along with prime attribute so bydefault it is 2NF because other attributes are cancelling the effect of Partial dependency of T.

20/9/18

Q. R(A B C D)
Key B B → C, C → A C → D

[C]+ = {C, A, D}.

B → C (BCNF)
C → A (2NF)
C → D (2NF) Highest normal form = 2NF

Q. A → C (1NF) key AB.
B → D (1NF)

Highest normal form = 1NF

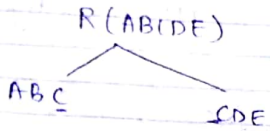
Q. R(ABCDE)
AB → D BCNF Key
C → A C → DA 1NF AB, BC, BE.
D → E 2NF
ABC → E 3NF.
BC → DC BCNF.

Highest normal form = 1NF

Q. R(ABCDE)

F = {AB → C
C → D
B → E }.

D1 = { ABC, CDE }.



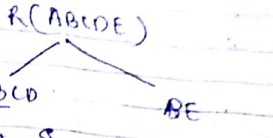
$[C]^+ = \{C, D\}$ (lossy).

↳ It is not covering any decomposition relation fully so it is lossy.

Q. $D_2 = \{ ABCD, BE \}$

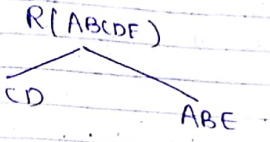
$D_3 = \{ ABCD, ABE \}$

$D_4 = \{ ABC, CD, DE \}$



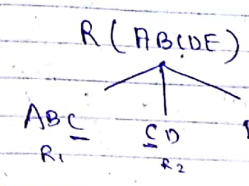
$(B)^+ = \{B, E\}$ (Lossless)

Covering BE



$[AB]^+ = \{A, B, C, D, E\}$.

↳ It is covering both the table (Lossless).



R_{12} $[C]^+ = \{C, D\}$ (lossless).

R_{12} and R_3 $[D]^+ = \{D\}$. (Lossy).

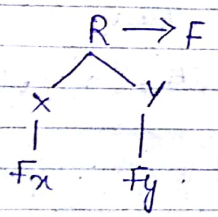
Combing R_2 and R_3 first

$(D)^+ = \{D\}$ (not covering any relation)

So, it is lossy in all the cases.

Preserving dependencies

When all the dependencies in the original relation is existing in the decomposition relation (taken together) then we can say dependencies are preserving



Condition $(F_x \cup F_y)^+ = F^+$

Eg $R(ABC)$ $F = \{A \to B, B \to C, C \to A\}$

$R_1(AB)$ $R_2(BC)$

$F_1 \{A \to B, B \to A\}$ $F_2 \{B \to C, C \to B\}$

So on combining $[F_1 \cup F_2]^+ = F^+$

Thus, the dependency is preserving

Eg $R(ABCD)$ $A \to B$, $B \to C$, $C \to D$

$R_3(CD)$ $R_1(AB)$ $R_2(BC)$

We will first find closure of that attribute that is only present in relation.

for, $R_1(AB)$ we will find closure of A and B only

$[A]^+ = \{A, B\}$ $[B]^+ = \{B\}$.

So dependency $= A \to B$ ✓

For $R_2(BC)$

$[B]^+ = \{B, C\}$. $[C]^+ = \{C\}$.

$B \to C$ ✓

For $R_3(CD)$

$[C]^+ = \{C, D\}$ $[D]^+ = \{D\}$.

$C \to D$ ✓

Thus $(R_1 \cup R_2 \cup R_3) = F_{original}$

so dependency is being preserved.

Q. R(ABCD)

$A \to B$
$B \to C$
$C \to D$
$D \to A$

$R_1(AB)$ $R_2(BC)$ $R_3(CD)$

$R_1(AB)$

$[A]^+ = \{A, B, C, D\}$

$A \to B$
$B \to A$

$R_2(BC)$

$[B]^+ = \{B, C, D, A\}$

$B \to C$
$C \to B$

$R_3(C,D)$

$[C]^+ = \{C, D, A, B\}$

$C \to D$
$D \to C$

$\begin{bmatrix} A \to B \\ B \to A \\ B \to C \\ C \to B \\ C \to D \\ D \to C \end{bmatrix} \Rightarrow \begin{Bmatrix} A \to B \\ B \to C \\ C \to D \\ D \to A \end{Bmatrix}$ all exists in original relation

∴ The decomposition follows BCNF (Doubt about 3NF).

Q. R(ABCDEF)

$AB \to CD$
$C \to D$
$D \to E$
$E \to F$

$D = \{AB, CDE, EF\}$.

AB $[A]^+ = \{A\}$
$[B]^+ = \{B\}$
$[AB]^+ = \{A, B, C, D, E, F\}$

CDE $[C]^+ = \{C, D, E, F\}$
$[D]^+ = \{D, E, F\}$
$[E]^+ = \{E, F\}$

$\begin{bmatrix} C \to D \\ C \to E \\ D \to E \end{bmatrix}$

∴ EF.

$E \to F$.

Finally we get, $\begin{Bmatrix} C \to D \\ C \to E \\ D \to E \\ E \to F \end{Bmatrix}$ which is not as original fd dependency.

So, it is not preserving dependencies

Q R(ABCDEG)

AB → C
AD → E
BC → A
AC → B
B → D
E → G

D = { ABC, ACDE, ADG }

ABC

[AB]+ = { A, B, C, D, E, G }

[B]+ = { B, D }

[BC]+ = { A, B, C, D, E, G }

[AC]+ = { A, B, C, D, E, G }

[AB → C
BC → A
AC → B]

ACDE

[AC]+ = { A, B, C, D, E, G }

[AD]+ = { A, D, E, G }

[AD → E
AC → D]

ADG

AD → E ; AD → G

F1 U F2 U F3

AB → C ✓
BC → A ✓
AC → B ✓
AD → E ✓
AC → D
AD → E
AD → G

AB → C
A → E
B → G

B → D and E → G doesn't exist in final relation.
So dependency doesn't preserves.

Minimal Cover (Read yourself).

— x —

28/9/18

JDBC (Java database connectivity)

Transactions

→ Concurrent users :- (independent of each other transaction)
→ Collection of operations that forms the logical unit of database processing is known as transaction.

→ Read only transaction
→ Write only transaction

There is a database variable x,
① Address of block where x is used
② Pick the block up and keep it main memory buffer.
Disk → Buffer.
Assuming that program variable is x and data variable is also x

In Read_item(x) :- The data variable will be read and will be stored in program variable.

In Write_item(x) :- The data variable will be first read and be stored in program variable, then changes may be made (as write operation) and again it will be stored in memory

Ti Begin T
Read (X)
X = X - A
Write (X)
Read (B)
B = B + A
Write (B)
End T

Suppose failure occur here

Considered as one single unit by compiler.

Suppose X = 1000 ; A = 50 ; B = 2000
X = X - A ; X = 950
Write (X) ; X = 950 (will be saved in main memory)
B = B + A ; B = 2050
Write (B) ; B = 2050 (will be saved in memory)

* The variable will be stored in disk.

Consistency of Database :- The database was consistent previous the transactions and it will be same state (or consistent) after transaction also.
Before the transaction X + B = 3000
after also it is 3000.

→ So database will be in inconsistent state.
This property is known as atomicity
If some transaction is occuring, it will occur completely or not at all.

Durability :- If transaction is completed successfully and if there occurs some failure, (assume power loss), then when again we will retrieve the data then it will give saved value from the disk.
This is known as durability.

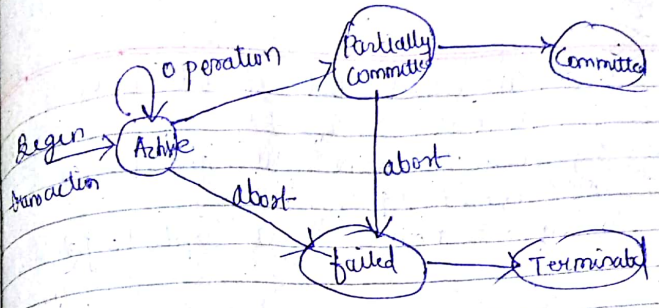
Isolation.

If there are 2 transaction T_i and T_j which started simultaneously, then both don't know any information about each other.

ACID :- Atomicity, Consistency, Isolation Durability.

There are concurrent transaction because :- ① Throughput max.
② Time minimum.
③ Resource utilization at maximum.

Eg :- If a user is only accessing a database for reading info. and another is using database for some transaction then both can similtaneously



Partially comitted :- When its last operation is completed.

Comitted :- When all the data is stored in the disk.

After fail we can restart or roll back the transaction or kill it.
But if we have to again do all transaction again then we kill it and start again.
If nothing can happen we will terminate the transaction.

Eg

T_1	T_2
R(X) 50	
X = X − N, 40	
	Read(X)
	X = X + M 55
write(X) 40 ← overwrite	W(X) 55
R(Y)	
Y = Y + N	
W(Y)	

Our consistent state should be X = 45 but we got 55.
Because X = X-N (transaction is not considered anywhere).
So this is known as loss update problem.

Eg.

T_1	T_2
R(X) 50	
X = X - N = 40 (N=10)	
W(X) = 40	
	R(X) 40
	X = X + M = 50 (M=10)
	W(X) = 50
R(Y)	

If this transaction fails somewhere here, so T_1 will roll back but T_2 will still have X value = 40, so this is known as dirty read problem.

* The space which we are leaving blw transaction is known as interleaving (i.e, transactions are occuring parallelly but the variable are not accessing parallelly as they are same)

T_1	T_2
R(X) = 50	
	R(X)
	X = X + M = (60)
	W(X)
R(X) = 60	

This problem is unrepeatable read →

when T_2 fails after T_1 read X, then T_2 roll back. So we have two values for X for T_1 : this is known as unrepeatable read (Now X will be 60 for T_1 which is inconsistent as T_2 has been roll backed).

[The solution of all these is serial order is seriability but there will be no concurrency & waiting time will be more.]

26/9/18.

S

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

This is serial transaction.

S'

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

This is non-serial or concurrent transaction

Schedule :- group of some transactions ($T_1, T_2 \cdots T_n$), but the order of transactions / operation in original transaction is same as schedule.

1st one is serial schedule.
2nd one is non-serial schedule or concurrent schedule.

Both of them are equivalent schedule as O/P is same.

Some non-serial schedule is equivalent to some serial schedule, then we can say that non-serial schedule is serializable schedule.

Eg

S	S'
R(X)	R(X)
X = X + 10	X = X * 1.1
W(X)	W(X)

These are not equivalent schedule.

Complete schedule :- If abort / commit statement is given along with the schedule then it is known as complete schedule.

T_i T_j
$i \neq j$

* The transaction dealing with different data items, then they will always be equivalent.
Problem is when we are dealing with same data items.

Eg:

T_1	T_2		T_1	T_2
R(A)		⟺		R(A)
	R(A)		R(A).	

Above order doesn't matter (as both are read operation).

Eg:-

T_1	T_2		T_1	T_2
R(A)		⟺	W(A)	
	W(A)			R(A)

If there is one W(A) write operation, the schedule won't be equivalent. (i.e, order matters).

Conditions for conflicting transactions.

1. Dealing with same data item.
2. One of the operations should be write.
3. There must be different transaction.

Eg:

T_1	T_2		T_1	T_2
W(A)		⟺		W(A)
	W(A)		W(A)	

This is not equivalent (i.e, order matters) as in first case T_2 transaction value will be saved and in second transaction T_1's A value will be saved to disk.
So in both the cases, values of A will be different.

If non-conflicting operations are being swapped to make a serial or non serial schedule (having same O/P) then they are non-conflicting equivalent.

Eg:-

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

If there are given n transactions there there will be n! serial transactions.

There will be more than n! concurrent transactions.

S'

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

S'

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

S'

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

S'

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

S'

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

S'

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

S'

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

S

T_3	T_4
R(B)	
	W(B)
W(B)	

S_1

T_3	T_4
R(B)	
W(B)	
	W(B)

S_2

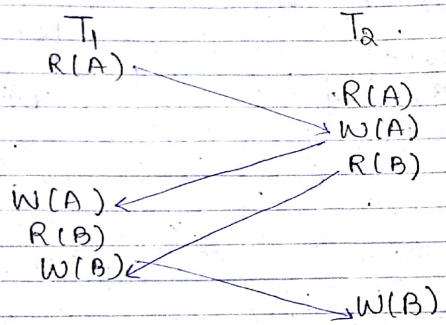
T_4	T_3
W(B)	
	R(B)
	W(B)

Both S_1 and S_2 are not conflicting equivalent to S.

As all the transactions in S are conflicting.

Some non-serial conflict schedule is equivalent to serial conflict schedule that is known as conflict serializable.

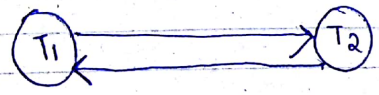
Ex



One approach to know whether is conflict equivalent is brute force (normal).

Another approach is precedence graph.

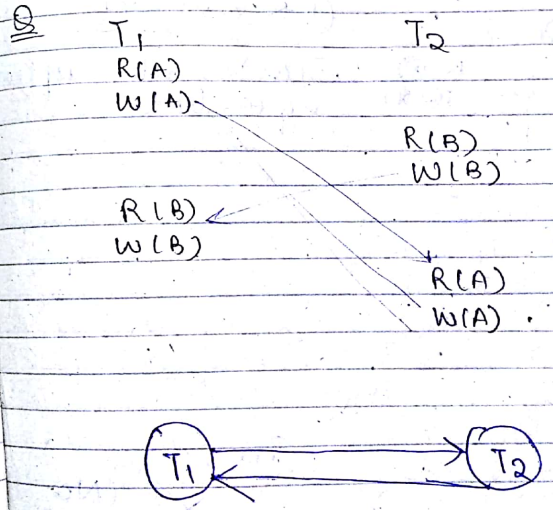
Precedence graph:



→ If cycle exist in graph then it is not conflict equivalent. (i.e, conflict occurs).

→ Edge in the graph will be drawn for RW, WR && RR operations (i.e, all conflicting operations).

Ex



27/9/18

Q. T_1 T_2

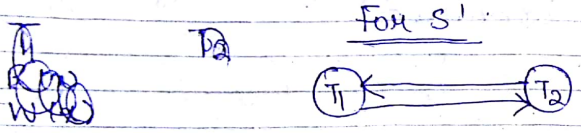
R(x)
W(x)
R(y)
W(y)

R(x)
W(x)

S'

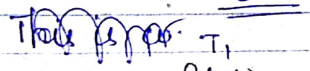
T_1	T_2
R(x)	
	R(x)
W(x)	
R(Y)	
	W(y)
W(Y)	

For S'



This is not conflict equivalent.

S''



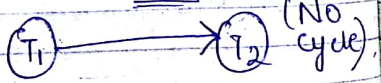
T_1 T_2

R(x)
W(x)

R(x)
W(x)

R(y)
W(y)

For S''



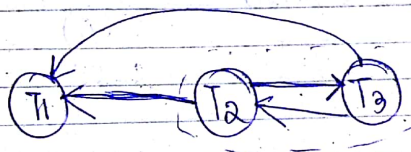
(No cycle)

This is conflict serializable.

Q. $R_2(A)\ R_3(C)\ W_3(A)\ W_2(A)\ W_2(B)\ W_3(C)\ R_1(A)\ R_1(B)\ W_1(A)\ W_1(B)$.

Subscript tells the no:- of transac. Here it is 3

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



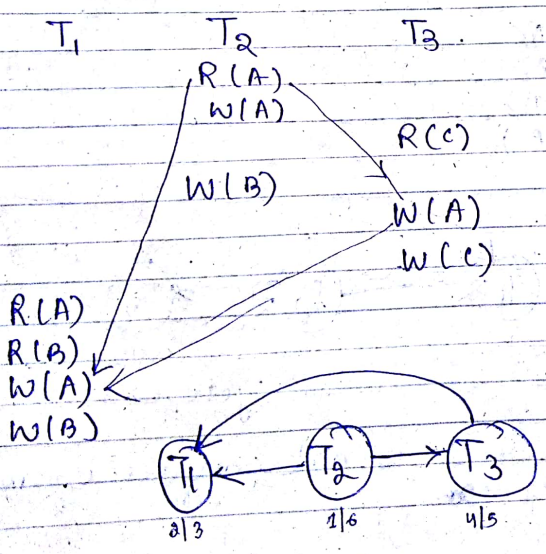
cycle exist.

So it is not conflict equivalent

If suppose it is not conflict equivalent, then for seriability there are 6 cases possible. (3!)
So, if we apply topological sorting on the precedence graph then we will get the exact order of seriali-zability.

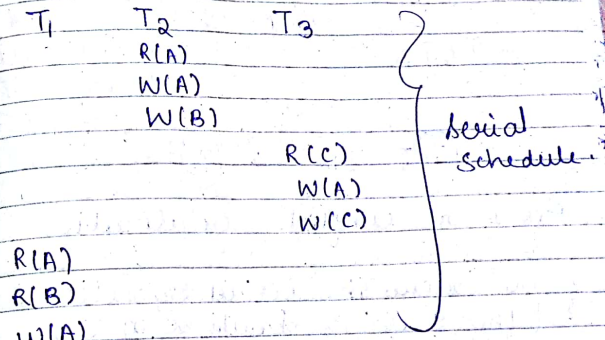
* Blind write :- Write before read

Q $R_2(A)$ $W_2(A)$ $R_3(C)$ $W_2(B)$ $W_3(A)$ $W_3(C)$ $R_1(A)$ $R_1(B)$ $W_1(A)$ $W_1(B)$

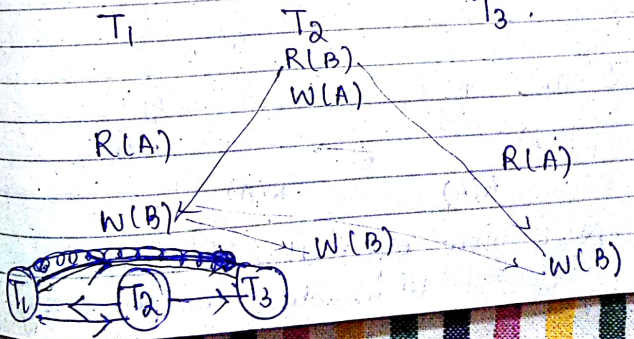


This is conflict equivalent.

T2 T3 T1 (Conflict serializable order)

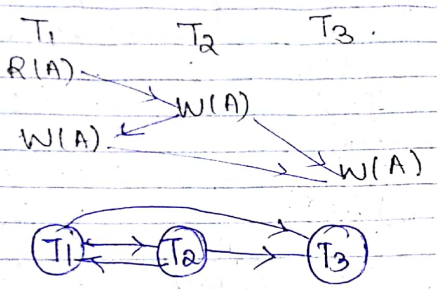


Q $R_2(B)$ $W_2(A)$ $R_1(A)$ $R_3(A)$ $W_1(B)$ $W_2(B)$ $W_3(B)$



Cycle exist. This is not conflict serializable.

Eg:-



This is not conflict serializable.

If we execute the serial execution of this then T_3 should be in last. (but here there is conflict b/w T_1 & T_2).

* If our graph is acyclic - then it is definitely conflict serializable but if cyclic it may or may not be conflict serializable.

View Serializability conditions:-

① Initial read.

S		S'	
T_i	T_j	T_i	T_j
R(A)			R(A)
	R(A)	≠	R(A)

Not view equivalent as two different transactions are performing initial read operation.

In S it is T_i and in S' it is T_j.

S		S'	
T_i	T_j	T_i	T_j
R(A)		≠	W(A)
	W(A)		R(A)

② Write read order (followed by)

Eg:-

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

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

This is not view equivalent → as T_1 is reading after writing of T_2 in S but in S' T_1 is reading after writing T_3.

③ Final updation.

Eg:-

T_1	T_2	T_3
	W(A)	
		W(A)
W(A)		

T_1	T_2	T_3
W(A)		
		W(A)
	W(A)	

In S final write is performed by 3.

but in S' final operation is performed by 2. So this is not view equivalent.

Q $R_2(B)\ W_2(A)\ R_1(A)\ R_3(A)\ W_1(B)\ W_2(B)$

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

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

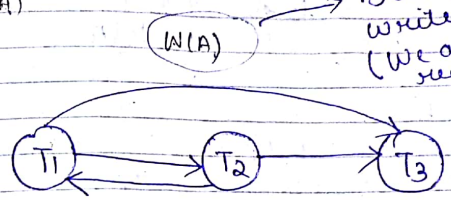
Here 2 condition is false as Read write update and Initial read also. So it is not view serializable.

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

Initial read (X)
Final update (X).
So it is not view serializable

22/9/18

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



Not conflict serializable.

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

(Serializability $T_1\ T_2\ T_3$)

View serializable. (✓).

★ If it is conflict serializable it will be view serializable

★ If it is cyclic graph, then if in whole schedule there is blind write then only it will be view serializable (i.e. first update also) Otherwise it won't be view ||

Q. T_1 T_2 T_3 T_4

R(A)

R(A)

R(A)

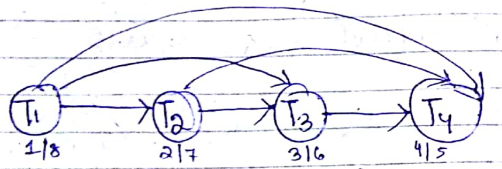
R(A)

W(B)

W(B)

W(B)

W(B)



This is conflict serializable and so it is also view serializable

Serial schedule will be $T_1 T_2 T_3 T_4$

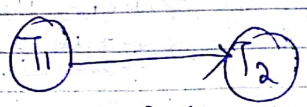
Q.

T_1	T_2	T_3	T_4
R(A)			
W(B)			
	R(A)		
	W(B)		
		R(A)	
		W(B)	
			R(A)
			W(B)

Total schedules that will be view serializable = 3! = 6.
As T_4 will always come at last as due to final update condition, rest are 3 transaction T_1 T_2 T_3 (Now write at read can be of any 4 but T_4 should come at last so we are left with 3 options) so we have 3! = 6 combinations possible.

Q $R_1(A)$ $W_1(A)$ $R_2(A)$ $W_2(A)$ $R_1(B)$ $W_1(B)$ $R_2(B)$ $W_2(B)$.

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



Conflict serializable and also view serializable

There are 2 serial schedules possible (as two transaction) But our question is equivalent to only 1 serial schedule. i.e., T_1 followed by T_2.

Q.

T_1 i	T_2 j
R(A)	
W(A)	
	R(A)
	commit.
R(B).	

If T_1 fails somewhere here then we have to rollback T_1 to ensure atmocity as T_2 is dependent on T_1 so we have to rollback T_2 but T_2 had commited so it can't be rollback so this is schedule is non-recoverable schedule.

If T_j reads a value written by T_i then the commit of T_i should come before the commit of T_j → then our schedule is recoverable.

T_1	T_2	T_3
R(A)		
R(B)		
W(A) ← depends		
	R(A)	
	W(A) ← dependent	
		R(A)

Abort.

→ If T_1 gets aborted then T_1 have to be rollbacked, T_2 is dependent on T_1 so T_2 should be rollbacked and T_3 also have to rollback. This is known as cascade rollback. This is a serious issue so it should be casacadeless.

→ Our schedule should be cascadeless for this commit of T_1 should be before the ~~R(A)~~ read of T_2 and commit of T_2 should be before read of T_3.

* Every cascadeless is recoverable.

Eg:-

T_1	T_2
R(A)	
W(A)	
commit.	
	W(A) / R(A)

If T_j is reading or overwriting a value written by T_i then commit of T_i should occur before reading / writing of T_2.
This is strict schedule.

* Every strict schedule is cascadeless and recoverable.

Q $R_1(X)$ $R_2(X)$ $W_1(X)$ $W_2(X)$
C_2 C_1

T_1	T_2
$R(X)$	
	$R(X)$
$W(X)$	
	$W(X)$
	C_2
C_1	

This is not strict schedule.
This is cascadless.
This is recoverable.

Q $R_1(X)$ $W_2(X)$ $W_1(X)$ $R_1(X)$
C_2 C_1

T_1	T_2
$R(X)$	
	$W(X)$
$W(X)$	
$R(X)$	
	C_2
C_1	

cascadeless.
Not strict.
This is recoverable.

03/10/18

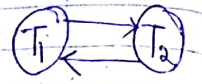
T_1	T_2	T_3
	$R(X)$	
		$W(X)$
		C
$W(X)$		
C		
	$W(Y)$	
	$R(Z)$	
	C	
$R(Y)$		
$R(Y)$		
C		

Not conflict serializable

This is a strict schedule so it is cascadeless and recoverable.

Q.

R(A)		
	R(A)	
	W(A)	
	R(B)	
W(A)		
R(B)		
W(B)		
	W(B).	



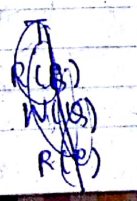
Not conflict serializable

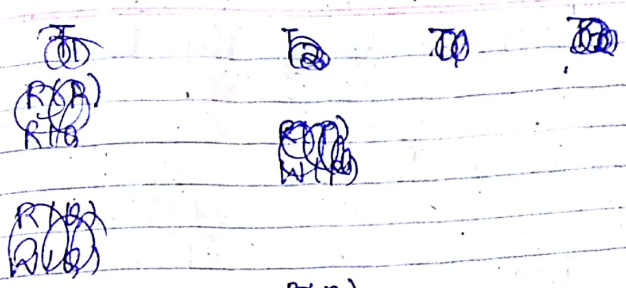
Q

T_1	T_2
R(P)	
R(Q)	
W(Q)	
	R(Q)
	R(P)
	W(P)

Any non-serial execution of this transaction is this CS or not.

No:- of serial schedule which can exist is 2. ($T_1 T_2$ or $T_2 T_1$).





As there is no non-serial schedule exist, then it is not conflict Serializable.

Q

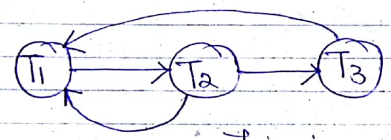
T_1	T_2
	R(X)
R(X)	
	R(Y)
W(X)	
R(Y)	
	W(X)
Abort	
	Abort

Not strict.

Cascadeless. (As there is no read in T_j after write in T_i and if T_j is reading a value written by T_i then commit of T_i occurs before reading of T_j → cascadeless)

Q

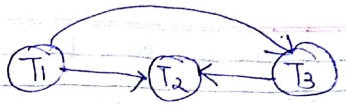
T_1	T_2	T_3
	R(Z)	
	R(Y)	
	W(Y)	
		R(Y)
		R(Z)
R(X)		
W(X)		
		W(Y)
		W(Z)
	R(X)	
R(Y)		
W(Y)		
	W(X)	



This is not conflict serializable

Q

T_1	T_2	T_3
R(X)		
	R(Y)	
		R(Y)
	W(Y)	
W(X)		
		W(X)
R(X)	W(X)	



Conflict serializable.

- T_1 T_3 T_2 is the serial order for the serial schedule.

Q R_1 (P, Q, R)
R_1 is having 2000 tuples.
R_2 (R, S, T)
R is FK
R_2 is having 2500 tupples.

Sol Maximum tupples in natural join = 2000

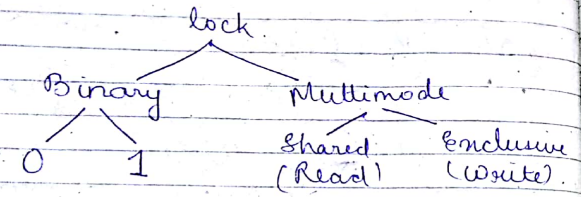
9/10/18

Concurrency control manager which checks whether concurrency exist or not.

Concurrency Protocol

① Locking Protocol

→ It sees the isolation property.
→ There is a variable lock.



T (request) → lock(X) → CCM → It will check the status of lock. If lock = 0 then request will be granted.

lock = 0 means it is unlocked.

lock = 1 means locked ie, there exist a transaction which is working on x. So then it goes (transaction) to wait state.

To store all this we need a table. So it stores a variable and transaction name.

→ Here also there is property of mutual exclusion. If one T is reading x, then T' is also requesting to read x. Then the request won't be granted.
So to overcome this we have multimode lock.

→ Shared lock is also known as read lock. There will only be read operation.
In exclusive both read write can be performed.

I_1 (exclusive operation)	I_2 (shared)
lock X (B)	lock S(A)
R(B)	R(A)
W(B)	unlock(A)
unlock(B)	lock X(B)
lock X(A)	R(B)
R(A)	W(B)
W(A)	unlock(B)
unlock(A)	

T ⟶ Q ⟶ mode M.

↳ both shared & exclusive.

	S	X
S	T	F
X	F	F

T → granted
F → not granted

↳ Compatibility matrix.

→ It is the work of CCM that the transaction/ lock should be in compatible mode. Then only access to variable will be granted. For this it takes help of compatibility matrix.

T_1	T_2	CCM (we generally don't show this part)
lock X (B)		
		grant X(B, T_1)
R(B) 200		
B = B - 50		
W(B)		
unlock(B)		
	lock S(A)	
	R(A)	grant S(A, T_2)
	unlock(A)	

T_1	T_2	CCM
	lock S(B)	
	R(B)	G1.
	unlock(B)	
	A+B	
lock X(A)		
		Grant
R(A)		
A = A + 50		
W(A) 150		
unlock(A)		

Here A+B should come 300 but our A+B at T_2 is coming out to be 250. So our value is inconsistent.

Non-serial schedule to remove inconsistency.

T_1	T_2	CCM
lock X(B)		
		grant
R(B)		
B = B - 50		
W(B)		
wait	lock S(A)	
	R(A)	
wait	lock S(B)	
lock X(A)		

so this is deadlock

→ We can handle data by roll backing transaction.
But we don't need incompatible data.

T_1 → lock X(B) T_2 → lock S(B)
This is not compatible and T_2 is waiting for T_1

T_2 - lock S(A) T_1 → lock X(A)
so T_1 is waiting for T_2 (so here there is cyclic wait and deadlock will occur).

T_1 —has→ S(A)
T_2 —req→ X(A)
T_3 —req→ S(A) Grant
T_1 → unlocked.

First T_2 is waiting for T_1 to release the lock. Then when T_1 release the lock then T_2 will wait for T_3. and this goes on. so T_2 is starving and this is known as starvation

2 phase locking protocol (2PL)

- growing / Expanding
  - They can obtain the lock not unlock
- shrinking.

All the commands before the first unlock is growing phase.

In shrinking phase we can only release the lock.

whenever first unlock comes that is known as ~~unlock~~ unlock point. After this point no lock command will come and the phase is known as Shrinking phase.

lock X(A)
R(A)
W(A)
lock X(B)
R(B)
W(B) ——→ unlock point.
unlock(A)
unlock(B)

(lock X(A) … W(B): growing phase; unlock(A), unlock(B): shrinking phase.)

So in this there is problem of starvation.

→ All the schedule that follow 2PL are Conflict serializable but vice-versa is not always true.

Whose unlock will appear first will be the first transaction of the serial schedule.

2PL

Static/conservative

Known previously which variables are to be used and lock is obtained previously (i.e. in future).

But this is not feasible practically as we need future knowledge and concurrency is hard to achieve.

Conservative strict

(We will not unlock the exclusive write lock till it transaction is committed or aborted)

lock X(A)
R(A)
lock S(B)
R(B)
W(A)
unlock(A)
lock X(A)
R(A)
W(A)
unlock(A)
lock S(A)
R(A)

If T_1 fails here then we have to rollback both T_1 & T_2.

Rigorous 2PL :-

We will not unlock the exclusive / write / read lock till the transaction is committed or aborted.
so here there is least concurrency

10/10/18

Lock conversions

T_1	T_2
$R(a_1)$	$R(a_1)$
$R(a_2)$	$R(a_2)$
	Display $(a_1 + a_2)$
$R(a_n)$	
$W(a_1)$	

We cannot execute this transaction concurrently. As in T_1 we need exclusive lock for a_1 and shared for a_1 in T_2. which is not compatible so we do lock conversion.

S(a_1)

S(a_2)

S(a_n)

upgrade S(a_1)

upgrade → can only be done in growing phase.
It is for converting shared to exclusive.

downgrade → converting exclusive to shared.

	T_1	T_2	T_3
	S(a_1)		
		S(a_1)	
		S(a_2)	
	unlock(a_1)		
		unlock(a_2)	

The serial order will be T_2 T_1 T_3 as unlocking point of T_2 comes first then T_1 and then T_3.

Timestamp Ordering Protocol.

Timestamp is a unique identifier which is issued by database.

We can maintain it in 2 ways:-
1. Logical counter.
2. System Time.

In logical counter, we are maintaing a variable (having some max. value). Now whenever a new Trans. will come, the variable will be incremented by 1. When max. Trans. will be arrive, then we have to reset the counter variable.
There is a disadvantage in this, that when at any time all the (max) no of transac. will enter, then it won't get time to reset. So we move towards. System time.

TS(T_i)

T_1	T_2	T_3	T_4
5	10	20	30

T_1 → Oldest transaction.
T_4 → youngest transac.

According to time-stamp serial order will be T_1 T_2 T_3 T_4.

Eg:-

10	20
T_1	T_2
R(A)	
	W(A)
R(B)	
	W(B)

Serial order in this case is T_1 and then T_2 as TS of T_2 > TS of T_1.

Eg

10	20
T_1	T_2
	W(A)
R(A)	
	W(B)
R(B)	

Serial order will be T_2 and then T_1 instead it should be T_1 and T_2 according to TS but T_2 is W(A) before T_1.

* So it is not necessary that TS will always help in deciding serial order.

R_TS(X) :- Read time stamp.
It is the largest time stamp value where transaction performs a succesful read operation on variable X.
It is the youngest transaction reading X.

Eg:-

10	5	30	20
T_1	T_2	T_3	T_4
R(A)			
	R(A)		
		R(A)	
			R(A)

R_TS(A) = 30 (As T_3 is youngest timestamp)

W_TS(X) Largest time stamp of the transaction executing succesful write operation.
⇒ Youngest transaction.

→ All schedules will be conflicting serializable in the basic Timestamp ordering protocol.

T —request→ R(X). (First we check the conditions of conflicts & then the all)

① WTS(X) > TS(T).

Eg:-

10	20
T_1 or T	T_2 or T'
	W(X)
R(X)	

Somewhere here T_1 might have read X.

Now somewhere T_2 fails and it gets rollback.
Then T_1 will have 2 values for X.
So T → R(X) cannot be performed.
(So this request won't be granted)

older trans → younger trans

2) WTS(X) < TS(T)

10	20
T	T'
W(X)	
	R(X)

This will get successfully executed. As T' is younger transaction, it will always read the value written by T. So there is no problem as such in previous case.

T request W(X).

3) RTS(X) > TS(T)

10	20
T_0	T_B'
	R(X)
W(X).	

As the new transaction has read the initial value. And W(X) of older transaction appears after. But T_B will perform operations on initial value but now X has been updated by T_1. So this transaction won't get executed. (So this case is wrong).

④ WTS(X) > TS(T)

10	20
T	T'
	W(X)
W(X)	

Older transaction will overwrite the value of X written by the younger transaction. Which is not possible as T' is an younger transaction as its time stamp is greater. So the request won't be granted.

Leaving conditions ③ and ④ all other cases will allow the write request or the write request will be granted.

Q $R_1(A)$ $R_2(B)$ $W_1(C)$ $R_3(B)$ $R_3(C)$ $W_2(B)$ $W_3(A)$.

	T_1	T_2	T_3
1.	10	20	30
2.	10	30	20

Check whether this is possible or not

① 10 20 30

T_1 T_2 T_3

R(A)

R(B)

W(C)

R(B)

R(C)

W(B) W(A).

Serial schedule of this.

T_1 T_3 T_2.

10 20 30

T_1 T_2 T_3

R(A)

W(C)

R(B)

R(C)

W(A)

R(B)

W(B)

The above schedule is not possible. And T_2 should be roll back.

② When $T_1 = 10$ $T_2 > 30$ $T_3 > 20$

Then there will be No such problem $RTS(X) > TS(T)$ doesn't exist. So this schedule is possible.

Advantages of Timestamp Ordering

① No deadlock.

② All the schedules are conflict serializable

③ C.S.

Drawback.

① Problem of starvation, recoverable and cascadeless may exist.

Strict timestamp:

① 10 20

T_1 T_2

W(A)

R(A) / W(A)

We have to delay T_2 till T_1 gets commited.

Thoma's write rule: (Modification in Timestamp Ordering).

② 10 20

T_1 T_2

W(A)

W(A)

In TS ordering we were rolling back T_1. But T_1 is writing value which is obsolete as it is an older transaction. So instead of rolling back.

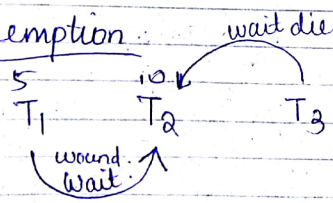
Deadlocks.

Cyclic wait → so deadlock.

Prevention of Deadlock

① Knowing the future operation of transaction, which is not practically possible.

② Preemption:



T_1 will wait till T_2 will release the lock.
We will forcefully release lock from T_2 and will give it to T_1.

If T_3 is requesting for resources held by T_2, then T_3 will wait for T_2.
(ie, T_3 will rollback).
So in both the cases, the younger transaction will be roll-backed.

Deadlock Detection.

① Wait for graph.

$T_i \longrightarrow T_j$.

If cycle → then deadlock will exist.

Victim selection:- To rollback which transaction.
We will rollback that transaction which is younger & have done the least update.

— × —

10/10/18

Tutorial-

Q R(STUV)
$R = R_1$ and R_2
$R_1 \cap R_2 = 0$

$S \to T$
$T \to U$
$U \to V$
$V \to S$

As R_1 and R_2 are disjoint, BCNF and 3NF doesn't support disjoint relation.
So highest NF = 2NF.

Q ① Select pid from R where class is AC and exist (select * from P where age > 65 and P.pid = R.pid);

P

pid	pname	age
0	S	65
1	R	66
2	B	67
3	A	69

R

pid	class	tid
0	AC	8200
1	AC	8201
2	SC	8201
5	AC	8203
1	SC	8204
3	AC	8024

Sol-

1 R 66
2 B 67
3 A 69

0 class is AC
1
5
3

1 & 3 → Ans

Q Book(title, price)
No two books have same price

① Select title from book as B where (Select count(*) from book as T where T.price > B.price) < 5

Book as B:

Title	price
A	100
B	200
C	300
D	400
E	500
F	600

Book as T:

Title	price	
A	100	5
B	200	4
C	300	3
D	400	2
E	500	1
F	600	0

B C D E F → will be the answer which is the 5 most expensive books

Q

Loan

Borrower	BM	Loan Amt
R	A	10000
S	B	5000
M	A	7000

Select count(*) from ((Select Borrower, BM from Loan as S) natural join (

① Create table wt same attribute name (error)

② AB | BD
1 2 | ϕ ϕ
2 1

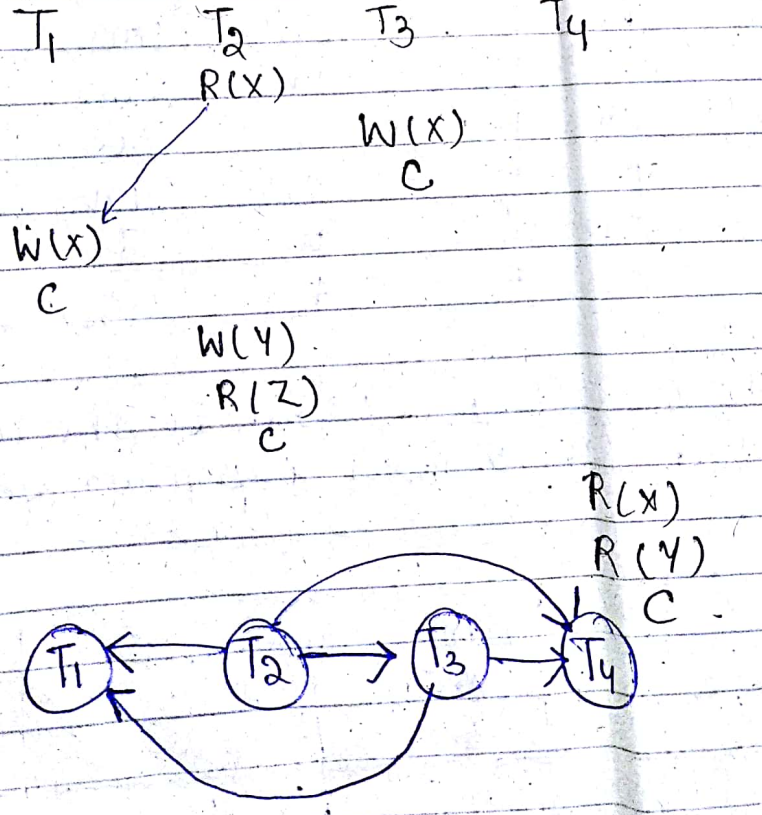
do cross product & natural join.
(It will give empty set).

Select Bn, from loan as T))

Sol- R A 10000
R A 7000
S B 5000
M A 10000
M A 7000

count(*) = 5.

Q.



Conflict serializable.