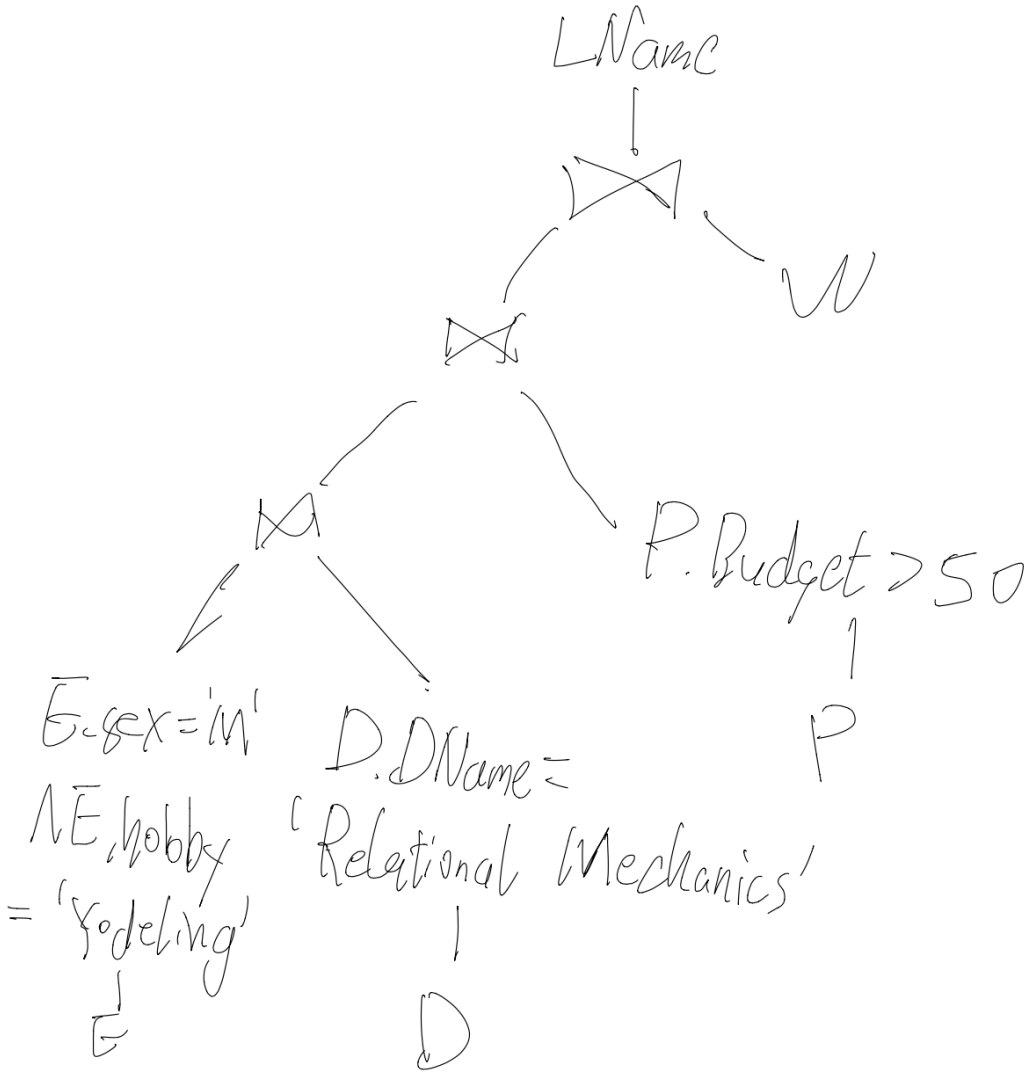
Databases hw4

Mou Zhang

(1)

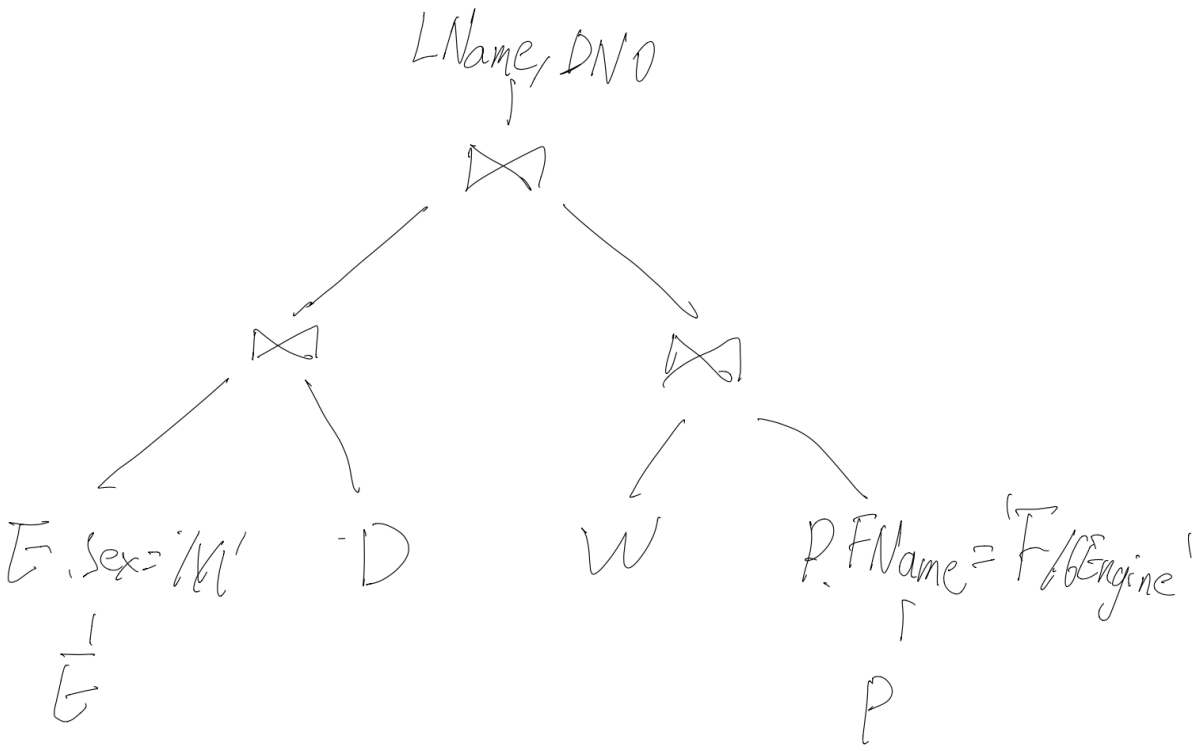
(1a)

$$\Pi_{Lanme}(\sigma_{Budget>50 \wedge Sex='M' \wedge hobby='Yodeling' \wedge Budeget>50}((Employee \bowtie_{DNO=DNO} Department \bowtie_{ESSN=ESSN} WorksON \bowtie_{PNUM=PNUM} Project)))$$



(1b)

$\Pi_{LName,DNO}(\sigma_{Fname='F16Engine' \wedge Sex='M'}(Employee \bowtie_{DNO=DNO} Department \bowtie_{SSN=SSN} WorksON \bowtie_{PNUM=PNUM} Project))$



(2)

(2a)

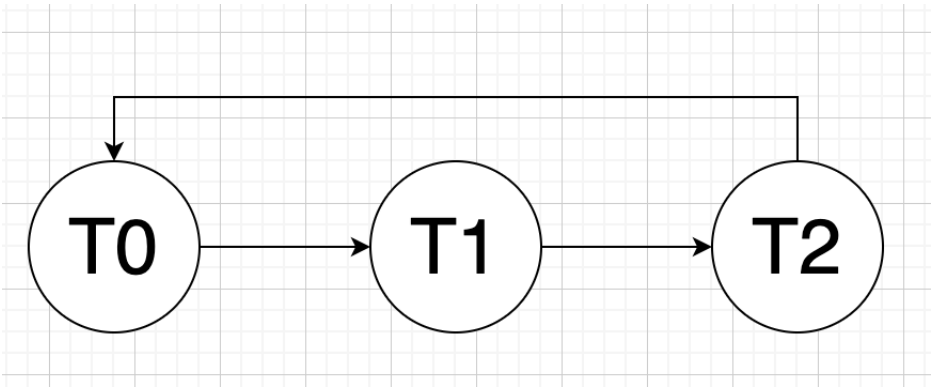
T_0	T_1
read_item(A) read_item(B) if A=0 then B :=B+1 write_item(B)	 read_item(B) read_item(A) if B=0 then A:=A+1 write_item(A)

(2b)

T_0	T_1
read_item(A)	
read_item(B)	
	read_item(B)
	read_item(A)
if A=0 then B :=B+1	
write_item(B)	
	if B=0 then A:=A+1
	write_item(A)

(3)

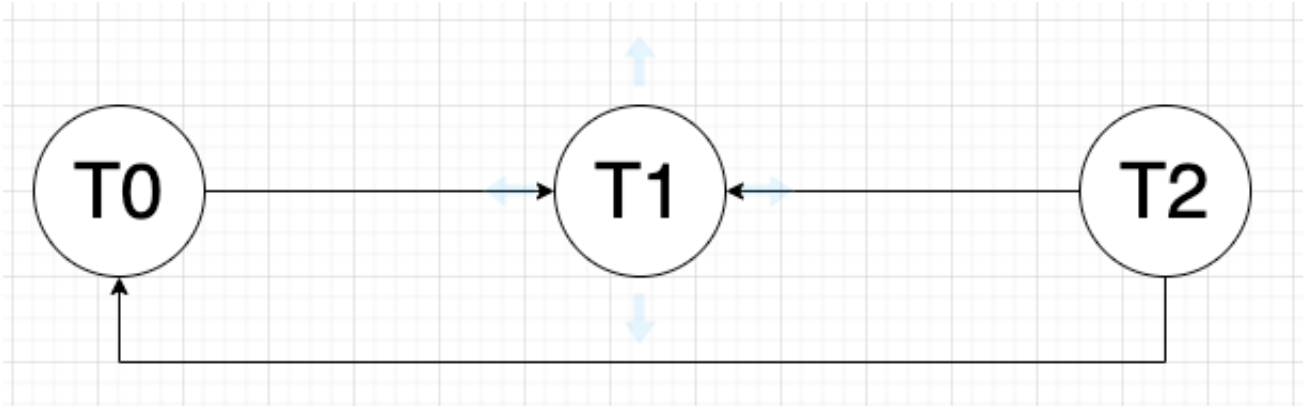
No, It is not conflict serializable.



As is shown in the precedence graph, there are cycles in the graph, means that it is not conflict serializable.

(4)

Yes, it is conflict serializable.



As is shown in the precedence graph, there are no cycles in the graph, So we can use topo sort to give the procedures an order: T2 -> T0 -> T1. The equivalent serial schedule is shown as below.

T_0	T_1	T_2
		read_item(Y) read_item(Z) write_item(Y) write_item(Z)
read_item(X) write_item(X) read_item(Y) write_item(Y)		
	read_item(Z) read_item(Y) write_item(Y) read_item(X) write_item(X)	

(5)

No, it is not conflict serializable.

(6)

Yes, it is conflict serializable.

The valid serial schedule should be like T4 -> T5 -> T1 -> T2 -> T3.

(7)

(a)

Process the query at the 'Boca' plant locally

(b)

Process the query at the New York office

(c)

Process the queries at plant location of Toronto, Edmonton, Vancouver, Montreal separetely

(d)

Process the query at the New York office.

(8)

(a)

First find the location of all plants containing machine number 1130 from MACHINE at the Armonk site. Then for each location found, let them find all employees info locally.

(b)

First find the location of all plants containing machine whosetype is 'Milling Machine' from MACHINE at the Armonk site. Then for each location found, let them find all employees info locally.

(c)

Find all machines whose PlantLocatiion is Almaden plant at the Armonk site.

(d)

There are two cases:

1. the number of contents in MACHINE table is bigger than the number of content in EMPLOYEE

Gather the EMPLOYEE table from each location to the Armonk site and calculate the join there.

2. the number of contents in MACHINE table is smaller than the number of content in EMPLOYEE

Send the MACHINE table separately to each location and every location only receives its own part of the MACHINE table with its locatimon. Then do the join locally at every location.

(9)

JOIN:

A	B	C	D	E
6	4	3	4	3
2	3	4	5	6
2	3	4	7	9

SEMIJOIN:

C	D	E
3	4	3
4	5	6

(10)

The hash function should fit the memory.

(11)

Adding all attribute of A which does not find a match with B to the hash table with its B set as NULL.

(12)

Q1: select * from stat_db;

Q2: select * from stat_db where name != target_name;

Q3: Q1 - Q2

(13)

R1 ∩ R2 = A

Since we have $A \rightarrow BC$, we know that $A \rightarrow ABC$, so $R1 \cap R2 \rightarrow R1$. Then according to the defination, this is a lossless-join decomposition.

(14)

A+ -> ABCDE

B+ -> BD

C+ -> C

D+ -> D

E+ -> ABCDE

(15)

$A \to BC$

$CD \to E$

$B \to D$

$E \to A$

(16)

Since we have, $B \to D$, $B^+ = (BD)$

(17)

The pseudotransitivity rule can be described as below:

If $\alpha \to \beta$ holds and $\gamma\beta \to \delta$ holds, then $\alpha\gamma \to \delta$.

Now we want to prove this rule. From Armstrong's axioms, first from $\alpha \to \beta$, we have $\alpha\gamma \to \beta\gamma$ (augmentation). Then since we have $\beta\gamma \to \delta$, we can combine this with $\alpha\gamma \to \beta\gamma$, and get $\alpha\gamma \to \delta$ (transitivity). Thus we have proved the pseudotransitivity rule.

(18)

select * from R as r1, R as r2 where r1.b = r2.b and r1.c != r2.c

This query should return an empty set

(19)

create assertion pseudotransitivity check(

no exists(select * from R as r1, R as r2 where r1.b = r2.b and r1.c != r2.c))

(20)

It is not a dependency preserving decomposition becuase you can not check $CD \to E$ and $B \to D$ without computing

$r1 \bowtie r2$.

(21)

R = {A, B, C, D}

F = {A $\rightarrow$ B, B $\rightarrow$ C, C $\rightarrow$ D}

key = {A}