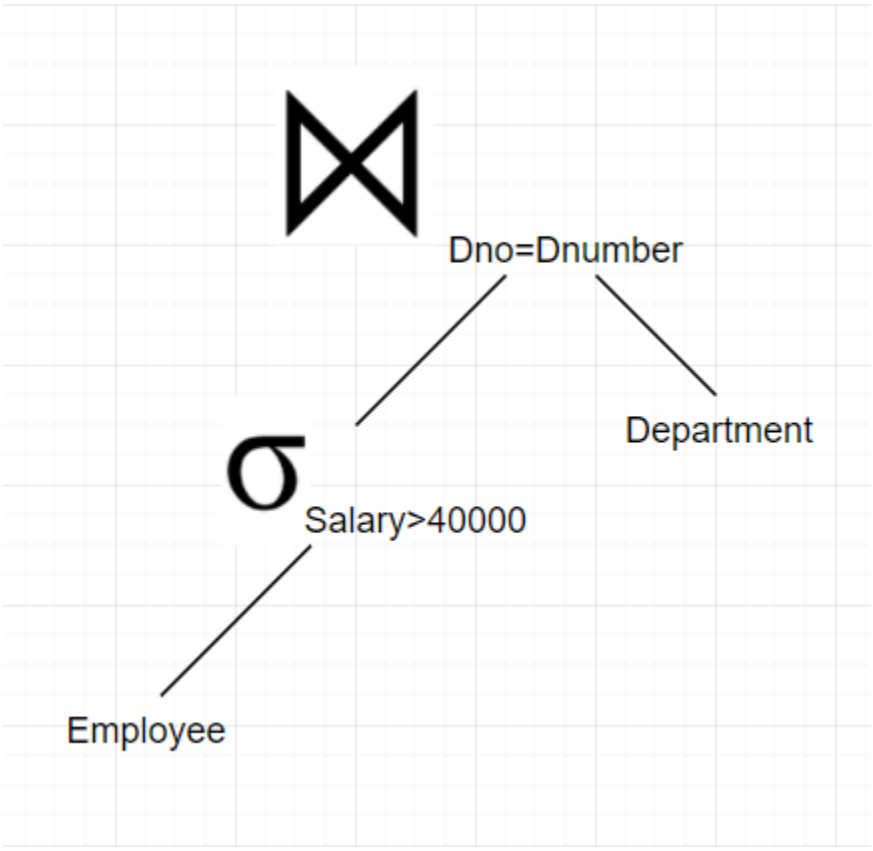
19.21) Query plan 1:



$\sigma_{\text{Salary>40000}} = \frac{500-400}{500} = \frac{1}{5}$

Cost of accessing index = 1 + 1/5 x 50 = 11

# of data blocks accessed = 1/5 x (# of rows)

=1/5 * 10000 = 2000

10000 rows are stored in 2000 blocks, this means that 500 rows can be stored per block. The temporary table will require 400 blocks to store the result, so the cost will be 400 blocks. If we use a nest loop join to join the temporary table with the department table, we have the following:

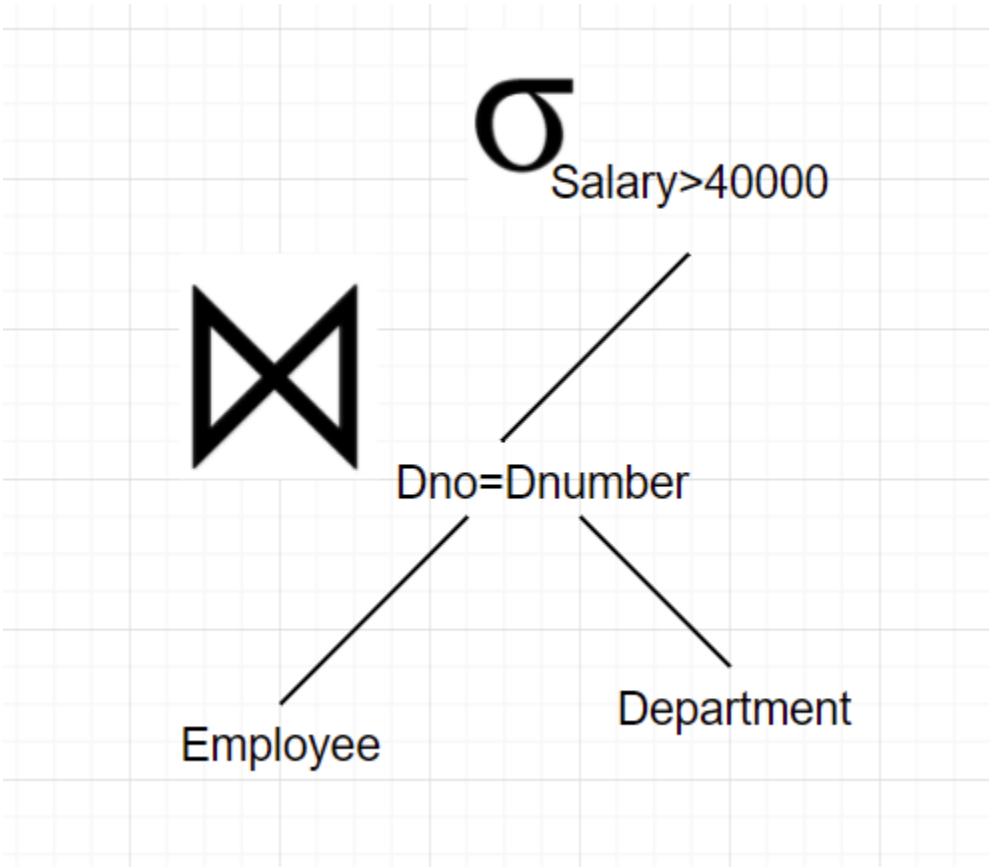
5+(5x 400) = 2005 block accesses

Total cost = Cost of accessing the index + Number of data blocks accessed + Number of blocks for TEMP table + Nested loop cost

=11+2000+400+2005

=4416 block accesses

Query Plan 2:



Assume a nested loop algorithm

# of data blocks accessed =2000

# of rows in DEPARTMENT = 50

Total cost = 50 + (50x 2000) = 100050 block accesses

20.14)

The condition does not change the final output unless the initial value of X >88. The outcome complies with the consistency rule that X<90 since X is not updated once it is larger than 90.

20.15)

From the initial data:

read_item(X);

X:=X+M

If X>90, then exit

else wite_item(X);

We can instead have transactions, T1 T2

read_item(X);

X:=X-N

read_item(X);

X:=X+M

write_item(X);

read_item(Y);

if X>90 then exit

else write_item(X);

Y:=Y+N;

If Y>90 then exit

else wite_item(X);

Y:=Y+N

If Y>90 then exit

else wite_item(Y);

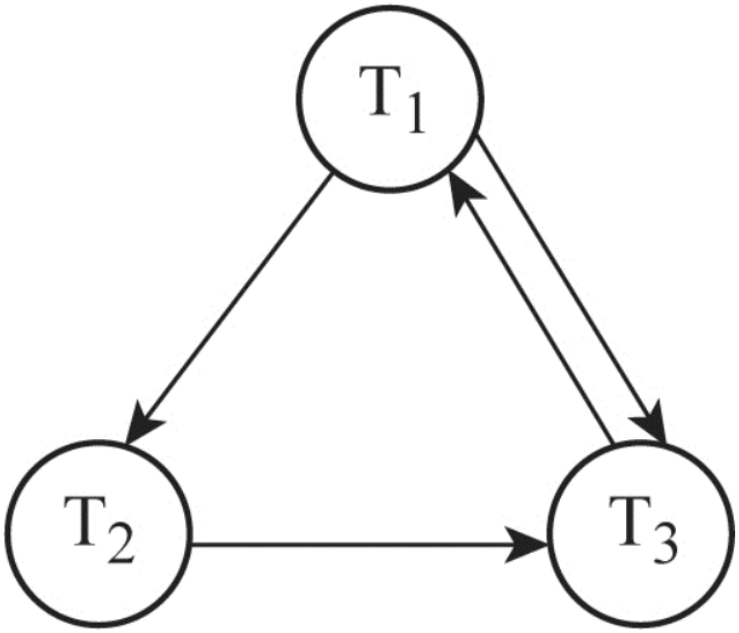
The condition does not change the final output unless the initial value of X> 88 or Y>88

The output keeps the consistency rule that X<90 and Y<90

20.22)

Given schedule: r1(X); r3(X); w1(X); r2(X); w3(X)

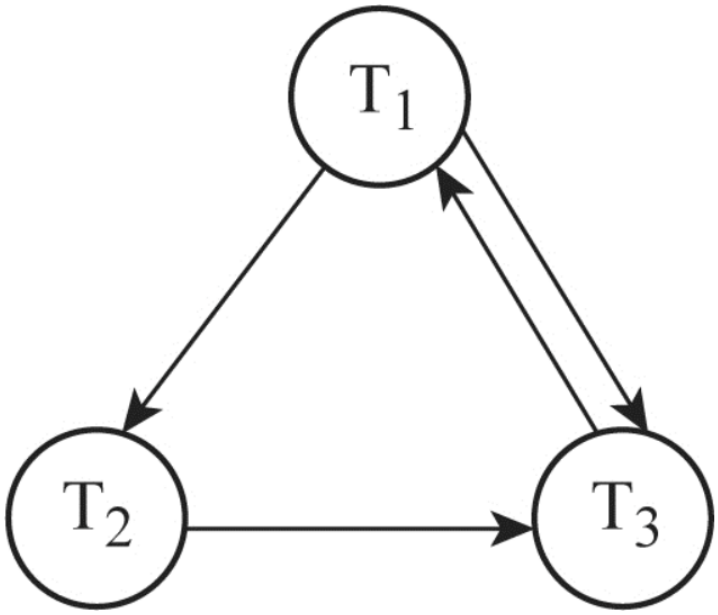
Conflict graph:



The conflict graph has cycle in T1-T3, the schedule is not serializable.

Given schedule: r1(X); r3(X); w3(X); w1(X); r2(X)

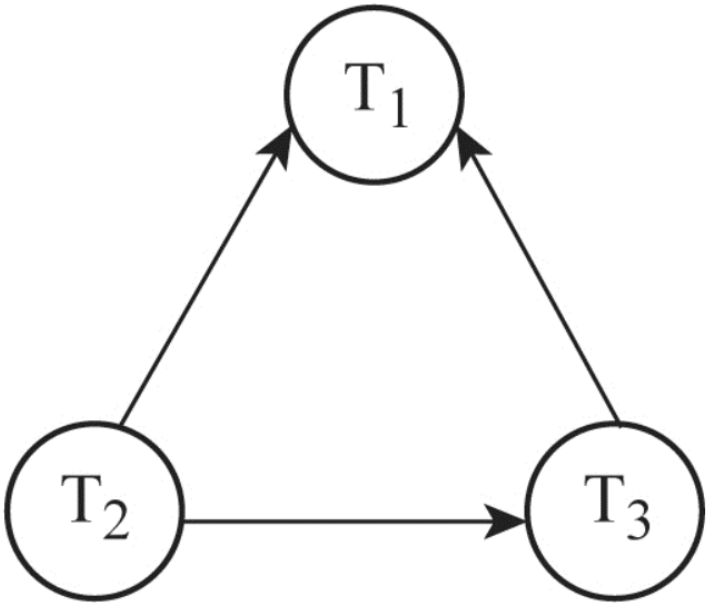
Conflict graph:



The conflict graph has cycle in T1-T3, the schedule is not serializable.

Given schedule: r3(X); r2(X); w3(X); r1(X);w1(X)

Conflict graph:

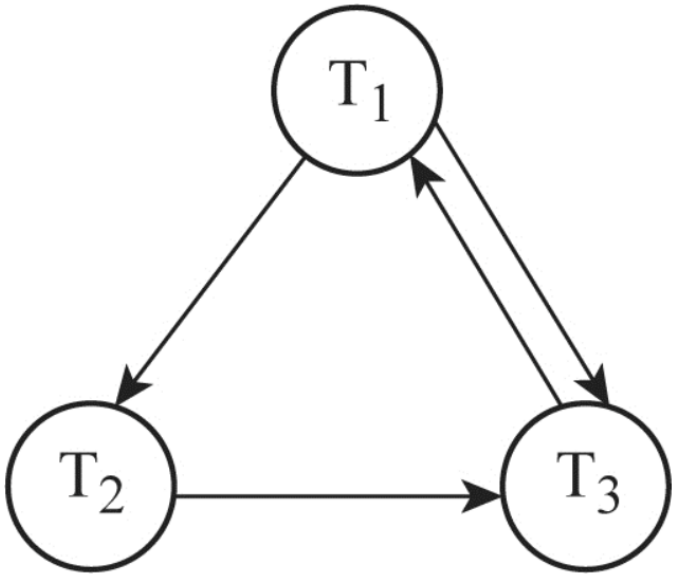


The graph has no cycles, so the schedule is serializable.

An equivalent schedule is: r2(X);r3(X);w3(X);r1(X);W1(X). The cycle is T2 -> T3 -> T1

Given schedule: r2(X);r2(X);r1(X);w3(X);w1(X)

Conflict graph:

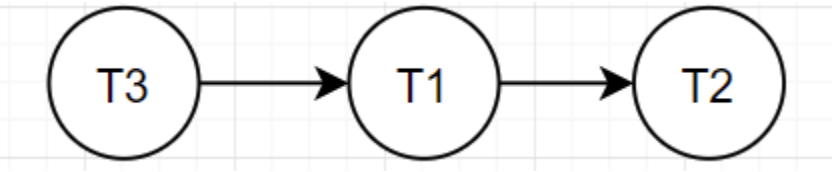


The conflict graph has cycle in T1-T3, the schedule is not serializable.

20.23)

Schedule S1: r1(x);r2(z);r1(x);r3(x);r3(y);w1(x);w3(y);r2(y);w2(z);w2(y)

Graph:



Schedule is serializable:

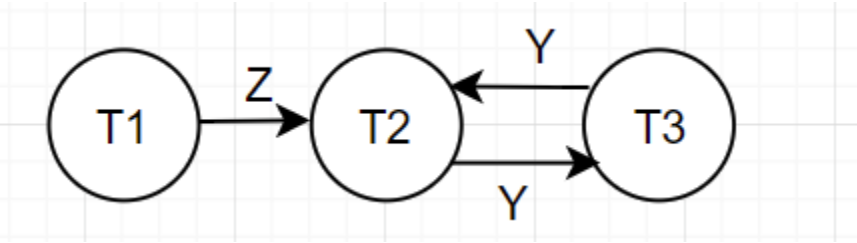
-T1 only reads X (r1(X)), r1 is not modified by T2 or T3.

-T3 reads X (r3(X)) before T1 modifies X

-T2 reads Y (r2(Y)) and writes (w2(Y)) only after T# writes (w3(Y))

S2: r1(X);r2(z);r3(x);r1(z);r2(y);r3(y);w1(x);w2(z);w3(y);w2(y)

Graph:



There is a cycle between T2 and T3, so it is nonserializable.

20.24)

Strict Schedule:

S3 is not strict because T3 must read X after C1, but T3 reads X before T1 writes X and T3 commits after T1

S4 is not strict because of the same reason above.

S5 is not strict because T3 reads X before T1 writes to X.

Cascadeless Schedule:

S3 is not cascadeless schedule because T3 reads X before T1 commits

S4 is not cascadeless schedule because T3 reads X before T1 commits

S5 is not cascadeless schedule because T3 reads X before T1 commits or T2 reads Y before T3 commits.

Recoverable Serializable:

Ci >Cj means Ci happens before Cj, Ai denotes abort Ti

If A1>C3>C2 then S3 is recoverable because rolling back T1 does not affect T2 or T3. If C1>A3>C2, then S3 is not recoverable because T2 reads Y after T3 wrote Y and T2 committed but T3 rolled back. If C1>C3>A3 then S3 is recoverable because roll back of T2 does not affect T1 and T3.

If A1>C2>C3, then schedule S3 is recoverable because roll back of T1 does not affect T2 or T3. If C1>A2>C3 then S4 is not recoverable.

If A1>C3>C2 then S5 is recoverable because T2 nor T3 writes to X. If C1>A3>C2 then S5 is not recoverable.