**Database 2 Endterm test 2021.12.09 12.15 Name: Lin Guohao**

**Neptun code: IW3XV9**

Solutions should be written into this file and sent to the following email address: [nikovits@inf.elte.hu](mailto:nikovits@inf.elte.hu)

**Exercise 1** (5 + 5 points)

**a)** The following is a sequence of **REDO**-log records written by transactions T, U and V:

<start T> <T,A,10> <start U> <U,B,20> <T,A,30> <start V> <U,D,80> <T,C,50>   
<U,B,40> <V,E,50> <U,D,50> <COMMIT V> <COMMIT U> <T,A,50> <END V> **<T,C,60>** (<COMMIT T> is not yet on disk)

Describe the action of the recovery manager, including changes to both disk and the log, if there is a crash and the last log record to appear on disk is: **<T, C, 60>**. You should give write(), output(), <log record>, FLUSH LOG actions in the appropriate order.

**Write the solution here:**

**WRITE(B,20) OUTPUT(B) WRITE(D,80) OUTPUT(D) WRITE(B,40)OUTPUT(B)),WRITE(D,50) OUTPUT(D)<END U>,FLUSHLOG**

**b)** The following is a sequence of **UNDO**-log records written by transactions T, U and V:

<start T> <T,A,10> <start U> <U,B,20> <T,A,30> <start V> <U,D,80> <T,C,50>   
<U,B,40> <V,E,50> <U,D,50> <COMMIT U> <abort V> <T,A,50> **<T,C,70>** (<COMMIT T> is not yet on disk)

Describe the actions of the recovery manager, including changes to both disk and the log, if there is a crash and the last log record to appear on disk is: **<T, C, 70>**. You should give write(), output(), <log record>, FLUSH LOG actions in the appropriate order.

**Write the solution here:**

WRITE<C,70> OUTPUT<C> WRITE<A,50>OUTPUT<A>WRITE<C,50>OUTPUT<C>WRITE<A,30>OUTPUT<A>WRITE<A,10>OUTPUT<A> <ABORT,T>,FLUSH LOG

**Exercise 2** (10 points)

Give the number of conflict-serializable schedules for the following pairs of transactions:

(Give some justification in several words too.)

a) **T1**: R1(A); W1(A); W1(C); R1(B); **T2**: R2(A); W2(B); W2(A);

b) **T1**: R1(A); W1(A); R1(B); W1(B); **T2**: R2(C); W2(C); W2(D);

c) **T1**: W1(A); R1(B); W1(B); R1(C); **T2**: R2(B); W2(B); R2(A);

**Write the solution here:**

a) R1(A); W1(A); W1(C); R1(B)||R2(A); W2(B); W2(A);

---------------------------

3!/(2!\*1!) = 3

R2(A); W2(B); W2(A)|| R1(A); W1(A); W1(C); R1(B)

1

Answer : 3+1 = 4

b)

R1(A); W1(A); R1(B); W1(B)|| R2(C); W2(C); W2(D)

-------------------------------------------------

7!/(4!\*3!) = 35

R2(C); W2(C); W2(D); R1(A); W1(A); R1(B); W1(B);

------------------------------------------------

7!/(4!\*3!) = 35

Answer = 35+35 = 70

c) W1(A); R1(B); W1(B); R1(C)||R2(B); W2(B); R2(A);

--------------------------

4!/(1!\*3!) = 4

R2(B); W2(B); R2(A)||W1(A); R1(B); W1(B);

=1

Answer = 4 + 1 = 5

**Exercise 3** (8 points)

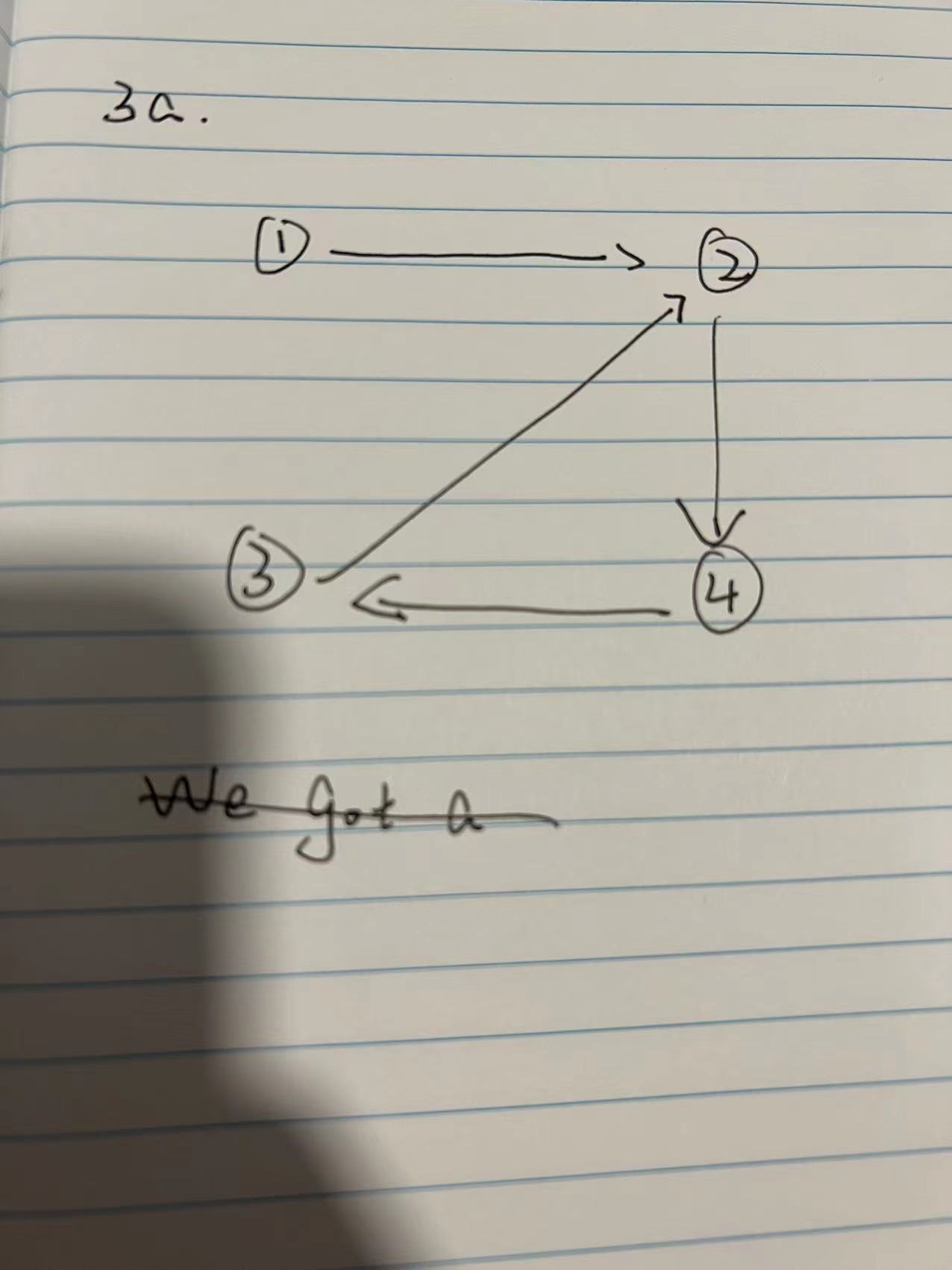
What is the precedence graph for the following schedule? Draw the graph with the following program: <https://app.diagrams.net/> . You can use **graph\_draw.svg** file on my homepage.

Is the schedule conflict-serializable? If so, what are all the conflict-equivalent serial schedules? (List only the transaction, not the operations.)

**a)** R1(B); R3(C); R4(D); R1(C); R2(B); W2(B); W2(C); W3(D); R4(B); W4(A);

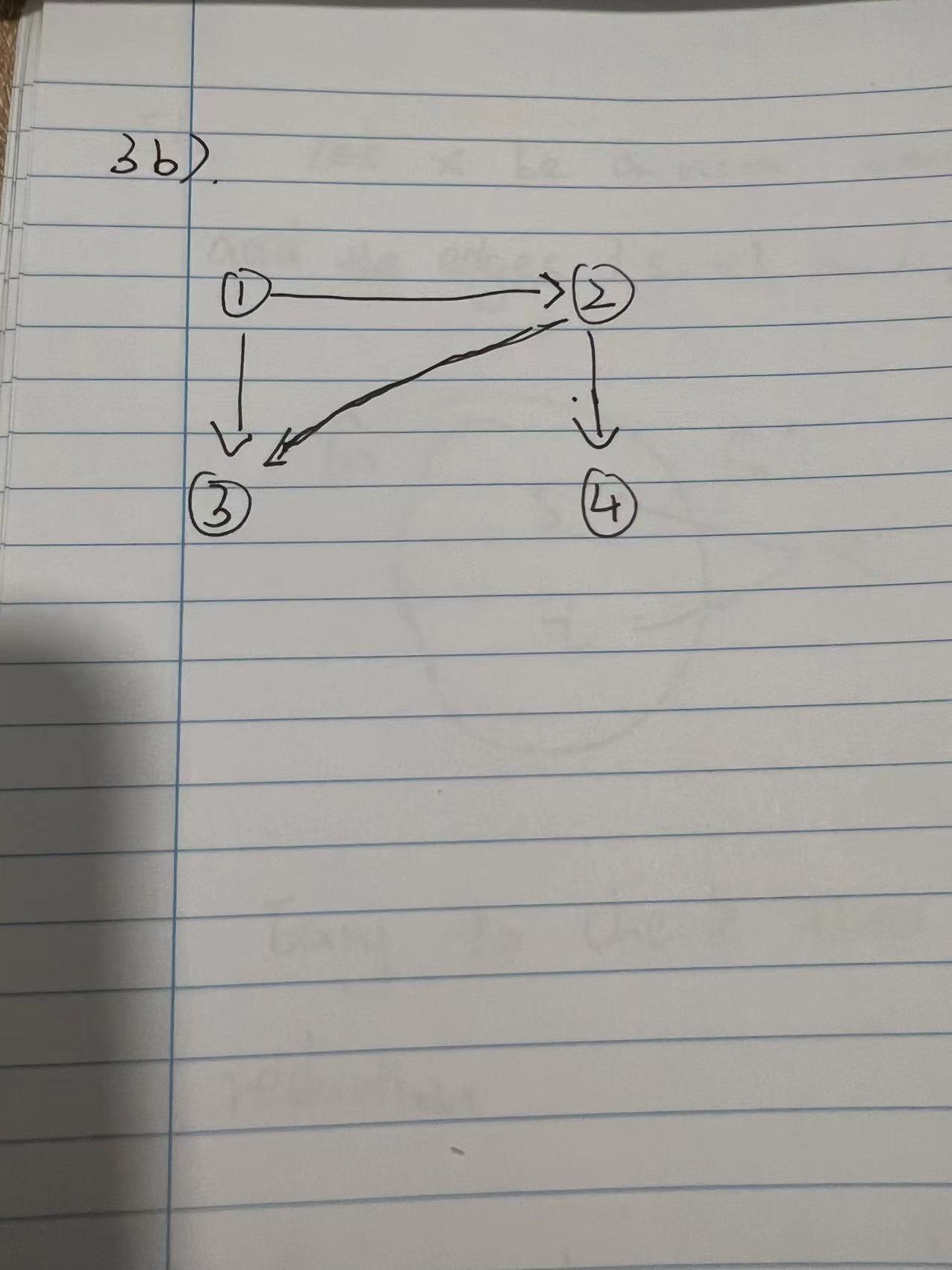
**b)** R1(A); W2(B); R1(C); R2(A); R4(B); W2(A); R3(B); W1(C); R3(C); W4(D);

3a)



We got a cycle, so the schedule is not a conflict-equivalent;

3b)



The schedule is conflict-serializable.

Conflict equivalent serial schedules are:

T1, T2, T4, T3

T1, T2, T3, T4

**Write the solution here:**

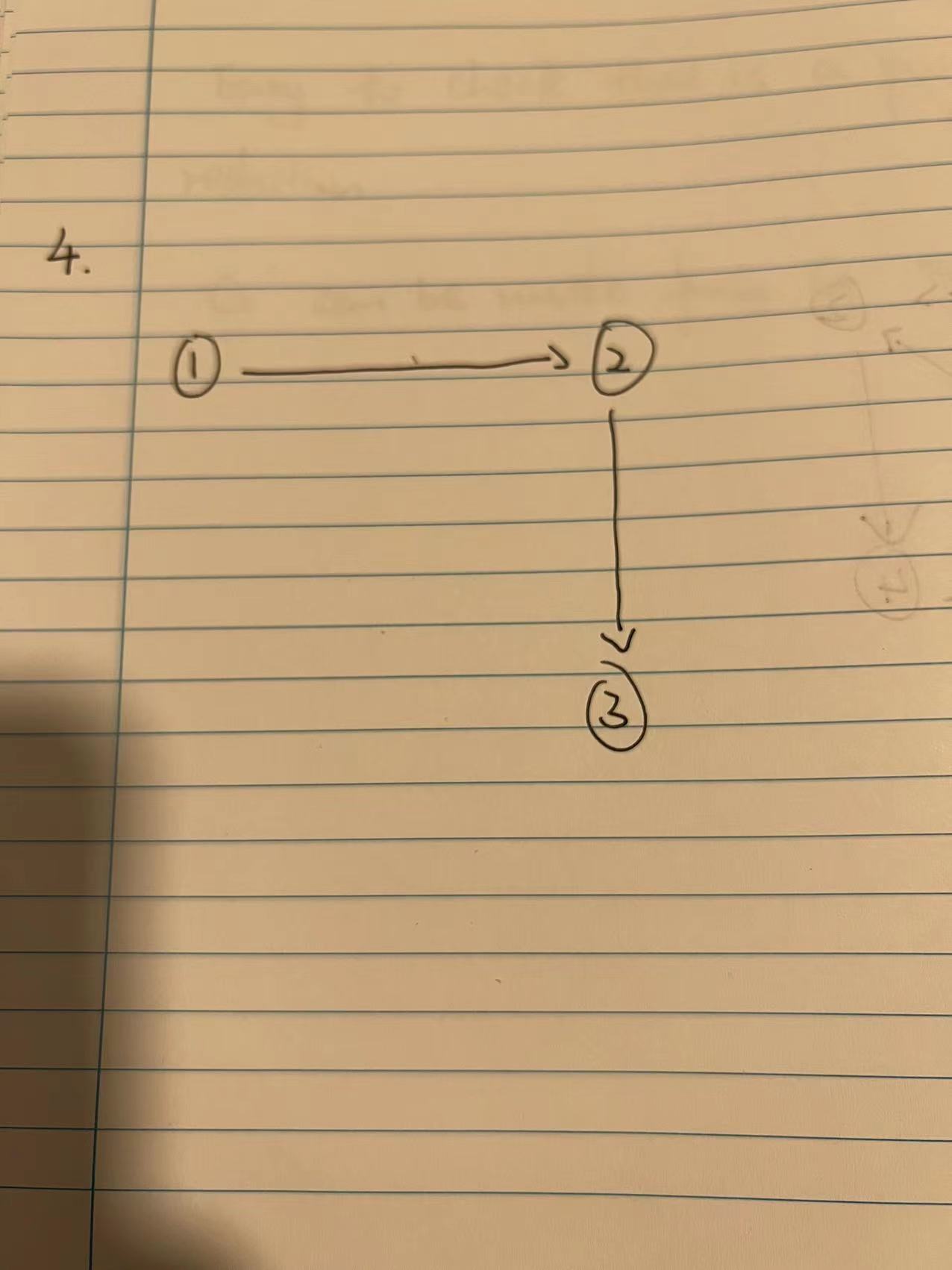
**Exercise 4** (8 points)

Here is a schedule with one action missing:

R1(B); W2(C); R2(A); ???; W1(D); R3(C); W2(B); W3(A)

Figure out which READ actions could replace the ??? and make the schedule not conflict-serializable. Find all possible solutions.

**Write the solution here:**



Three READ actions can replace ??? to make the schedule not conflict-serializable

R3(B) R2(D) R3(D)

**Computer Exercises**

**Exercise 5** (8 points)

Send in a separate text file the source code of **print\_histogram** procedure specified in **test2\_compulsory.txt** file, run your procedure with the following parameters and send the output as well: CALL print\_histogram('nikovits','test2','yr');

The owner of the following tables is NIKOVITS.

PRODUCT(prod\_id, name, color, weight)

SUPPLIER(supl\_id, name, status, address)

PROJECT(proj\_id, name, address)

SUPPLY(supl\_id,prod\_id,proj\_id,amount,sDate)

**Exercise 6** (2 x 4 points)

Give the sum amount of products where the color of the product is 'piros' (color = 'piros') and the **address of the project** is 'Pecs'. Give hints in order to get the following execution plans:

1. In the execution plan **all joins should be SORT-MERGE** **join**, and **no index** should be used for any table.
2. In the execution plan **all joins should be NESTED LOOP join**, and **one index** should be used for the product table.

Send the SQL queries, the result of the query, and the execution plans.

**Copy the solution here:**

**a)**

select /\*+ no\_index(s) no\_index(p) no\_index(pj) use\_merge(s p pj) \*/ sum(s.amount)amount from NIKOVITS.supply s , NIKOVITS.product p , NIKOVITS.project pj

where s.prod\_id = p.prod\_id and s.proj\_id =pj.proj\_id and p.color = 'piros' and pj.address = 'Pecs' ;

SELECT STATEMENT + +

SORT + AGGREGATE +

MERGE JOIN + +

SORT + JOIN +

MERGE JOIN + +

SORT + JOIN +

TABLE ACCESS + FULL + NIKOVITS.PRODUCT

SORT + JOIN +

TABLE ACCESS + FULL + NIKOVITS.SUPPLY

SORT + JOIN +

TABLE ACCESS + FULL + NIKOVITS.PROJECT

b)

select /\*+ no\_index(s) no\_index(p) index(pj) use\_nl(p s pj) \*/ sum(s.amount)amount from NIKOVITS.supply s , NIKOVITS.product p , NIKOVITS.project pj

where s.prod\_id = p.prod\_id and s.proj\_id =pj.proj\_id and p.color = 'piros' and pj.address = 'Pecs' ;

SELECT STATEMENT + +

SORT + AGGREGATE +

NESTED LOOPS + +

NESTED LOOPS + +

NESTED LOOPS + +

TABLE ACCESS + FULL + NIKOVITS.PRODUCT

TABLE ACCESS + FULL + NIKOVITS.SUPPLY

INDEX + UNIQUE SCAN + NIKOVITS.PROJ\_ID\_IDX

TABLE ACCESS + BY INDEX ROWID + NIKOVITS.PROJECT

**Exercise 7** (8 points)

Give a SELECT statement which has the following execution plan (**owner of tables is NIKOVITS**)

SELECT STATEMENT + +

SORT + GROUP BY +

CONCATENATION + +

TABLE ACCESS + BY INDEX ROWID BATCHED + NIKOVITS.SUPPLY

INDEX + RANGE SCAN + NIKOVITS.SUPPLY\_PROJ\_IDX

TABLE ACCESS + BY INDEX ROWID BATCHED + NIKOVITS.SUPPLY

INDEX + RANGE SCAN + NIKOVITS.SUPPLY\_SUPPLIER\_IDX

Send the SQL query and execution plan. If your solution is similar to the above, you get subpoints.

**Copy the solution here:**

SELECT/\*+ use\_concat index(s) index(p) \*/SUM(s.amount)

FROM NIKOVITS.SUPPLY s, NIKOVITS.PROJECT p

WHERE p.address = 'Pecs' or s.amount between 0 and 5

ORDER BY 1 DESC;

SELECT STATEMENT + +

SORT + AGGREGATE +

CONCATENATION + +

MERGE JOIN + CARTESIAN +

TABLE ACCESS + FULL + NIKOVITS.PROJECT

BUFFER + SORT +

TABLE ACCESS + FULL + NIKOVITS.SUPPLY

MERGE JOIN + CARTESIAN +

TABLE ACCESS + FULL + NIKOVITS.PROJECT

BUFFER + SORT +

TABLE ACCESS + FULL + NIKOVITS.SUPPLY