Universidade Federal do Espírito Santo
Centro Tecnológico
Departamento de Informática

Banco de Dados
Prof. Rodrigo Laiola Guimarães

Instruções gerais: O BOCA é um sistema de correção automática de exercícios que verifica se o resultado gerado pelo seu programa satisfaz casos de teste pré-definidos. Portanto, é necessário seguir estritamente os formatos especificados na questão. Lembre-se que os exemplos dados servem para facilitar o entendimento e podem não cobrir todos os casos de teste que serão usados.

O **teste de equivalência por conflito (serialização por conflito)** examina apenas as operações `read_item` e `write_item` em um *schedule* para construir um **grafo de precedência** (ou **grafo de serialização**), o qual é um **grafo direcionado** $G = (N, E)$ que consiste em um conjunto de nós $N = \{T_1, T_2, ..., T_n\}$ e um conjunto de arestas direcionadas $A = \{a_1, a_2, ..., a_n\}$. Existe um nó no grafo para cada transação T_i no *schedule*. Cada aresta e_i no grafo tem a forma $(T_j \rightarrow T_k)$, $1 \leq j \leq n$, $1 \leq k \leq n$, onde T_j é o **nó inicial** de a_i e T_k é o **nó final** de a_i. Tal aresta do nó T_j ao nó T_k é criada pelo algoritmo se uma das operações em T_j aparecer no *schedule* antes de alguma *operação de conflito* em T_k.

Algoritmo.

1. Para cada transação T_i participante no schedule S, crie um nó rotulado com T_i no grafo de precedência;
2. Para cada caso em S onde T_j executa um `read_item(X)` depois de T_i executar um `write_item(X)`, crie uma aresta $(T_i \rightarrow T_j)$ no grafo de precedência;
3. Para cada caso em S onde T_j executa um `write_item(X)` após T_i executar um `read_item(X)`, crie uma aresta $(T_i \rightarrow T_j)$ no grafo de precedência;
4. Para cada caso em S onde T_j executa um `write_item(X)` após T_i executar um `write_item(X)`, crie uma aresta $(T_i \rightarrow T_j)$ no grafo de precedência;
5. O schedule S é serializável se, e somente se, o grafo de precedência não tiver ciclos.

Se houver um ciclo no grafo de precedência, o *schedule* S não é serializável (conflito); se não houver ciclo, S é serializável. Um ciclo em um grafo direcionado é uma **sequência de arestas** $C = ((T_j \rightarrow T_k), (T_k \rightarrow T_p), ..., (T_i \rightarrow T_j))$ com a propriedade de que o nó inicial de cada aresta — exceto a primeira aresta — é o mesmo que o nó final da aresta anterior, e o nó inicial da primeira aresta é o mesmo que o nó final da última aresta (a sequência começa e termina no mesmo nó).

Escreva uma consulta que verifica se o *schedule* é ou não serializável de acordo com o algoritmo descrito.

Entrada:
Considere a existência da tabela **`Schedule`**, na qual cada linha representa a chegada de uma operação pertencente a uma dada transação (o número de transações presentes no *schedule* pode variar). A tabela possui 4 colunas: a primeira representa o tempo de chegada (`time`), a segunda o identificador da transação (`#t`), a terceira a operação (`read_lock` : bloqueio compartilhado para leitura de um item, `write_lock` : bloqueio (exclusivo) para escrita/gravação de um item, `unlock` : desbloqueio de um item, `read_item` : leitura de um item, `write_item` : escrita de um item, `commit` : confirmação ou `rollback` : aborto/*rollback*) e a quarta o item de dados (atributo) que será bloqueado/desbloqueado/lido/escrito (quando aplicável). As linhas da tabela estão ordenadas logicamente pelo valor na primeira coluna, que indica o carimbo (rótulo) de tempo (*timestamp*) de chegada (quanto menor o valor, mais antiga a operação).

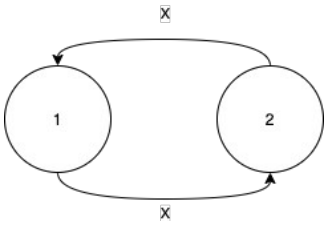
Saída:
A saída deve ser uma tabela contendo uma coluna chamada `RESP` com o valor `1`, se for serializável; caso contrário, `0`.

Exemplo 01

time	#t	op	attr
1	1	read_item	X
2	2	read_item	X
3	2	write_item	X
4	1	write_item	X
5	2	commit	-
6	1	commit	-

Exemplo 02

time	#t	op	attr
7	3	read_item	X
8	3	read_item	Y
9	4	read_item	X
10	3	write_item	Y
11	4	commit	-
12	3	commit	-



Saída

0

Saída

1