Instituto Superior de Engenharia de Lisboa
Área Departamental
de Engenharia de Electrónica e Telecomunicações e de Computadores
Licenciatura Engenharia Informática e de Computadores (2º Sem 2018/2019)

Sistemas de Informação II - 1.º teste (2019/06/18) Duração: 2:30 horas

Docentes: Afonso Remédios e Walter Vieira

---

1) Numa empresa, para a gestão de faturas foram criadas as seguintes tabelas:

Considere o modelo físico criado pelo código seguinte:

```
create table equipas(id int primary key,
                     descr varchar(50) not null
                    )

create table campeoes(id int not null references equipas,
                      ano numeric(4) primary key,
                      pontos int not null,
                     )
```

a. Sabendo que em alguns anos pode não haver campeão, implemente em TSQL a função **anosSemCampeao** que, recebendo no primeiro parâmetro um primeiro ano e no segundo parâmetro um segundo ano, devolve a tabela de todos os anos no intervalo correspondente aos parâmetros nos quais não houve campeão.

b. Usando a função implementada na alínea a), construa o procedimento armazenado **insAnosSemCamp** que recebe como parâmetros dois anos (@a1 e @a2) e procede da seguinte forma:

- Se ainda não existir em equipas um registo com id igual -1, insere um registo com esse **id** e **descr** '**' ;
- Insere em campeões todos os anos no intervalo de anos passado ligados à equipa -1 e com zero pontos.

c. Altere o código que desenvolveu na alínea b) de forma a que a inserção da equipa com **id** igual a -1 seja realizada por um gatilho definido sobre a tabela campeões. Deve apresentar também o código do gatilho.

---

2) Considere a tabela seguinte com os registos indicados:

```
create table t(i int primary key, j int)
insert into t values (1,1)
insert into t values (2,2)
```

Considere também 4 transações que executam os códigos seguintes:

`set transaction isolation level ?`
`begin tran --T1.a`
`  update t set j = j+10 where i=2--T1.b`
`  update t set j = j+20 where i=1--T1.c`
`commit--T1.d`
	`set transaction isolation level ?`
	`begin tran –T2.a`
	`  update t set j = j+10 where i>0--T2.b`
	`commit--T2.c`

`set transaction isolation level ?`
`begin tran --T3.a`
`  declare @x int`
`  select @x=j from t where i=2--T3.b`
`  update t set j=@x-1000 where i=2--T3.c`
`commit --T3.d`
	`set transaction isolation level ?`
	`begin tran --T4.a`
	`  declare @x int`
	`  select @x=j from t where i=2--T4.b`
	`  update t set j=@x+1000 where i=2--T4.c`
	`commit--T4.d`

a) Diga qual é o resultado da execução concorrente de **T1** e **T2** para cada um dos níveis de isolamento da norma ISO se o escalonamento for <T1.a,T1.b,T2.a,T2.b,T1.c,T1.d,T2.c>. Se para um dado nível de isolamento o escalonamento não for possível diga qual é o resultado para um escalonamento que mantenha o mais possível a ordem inicial do escalamento indicado. Assuma que ambas as transações correm com o mesmo nível de isolamento.

b) Idem para a execução concorrente de **T3** e **T4** se o escalonamento for <T3.a,T3.b,T4.a,T4.b,T3.c,T3.d,T4.c,T4.d>

c) Idem para a execução concorrente de **T3** e **T2** se o escalonamento for <.T3.a,T3.b,T2.a,T2.b,T3.c,T3.d,T2.c>

---

3) Responda às seguintes questões:

a) Um escalonamento estrito é sempre *cascadeless*?
b) Na norma ISO, em que condições é possível ocorrerem *deadlocks* com nível de isolamento *read committed*?
c) Na norma ISO, em que condições é possível ocorrerem *deadlocks* com nível de isolamento *read uncommitted*?

---

4) Considere uma base de dados com a seguinte tabela:

```
create table Conta(numero int primary key, cliente varchar(50), saldo real not null)
```

Nessa BD também foram implementados os seguintes procedimentos armazenados:

```
create proc debitar @num int, @valor real
as
begin set transaction isolation level repeatable read
    begin tran
      if not exists (select * from contas where numero = @num)
         begin rollback; return -1; end
      update contas set saldo = saldo - @valor where numero = @num and saldo >= @valor
      if @@rowcount = 0 begin rollback; return -2; end
  commit
  return 0
end

create proc creditar @num int, @valor real
as
begin set transaction isolation level repeatable read
    begin tran
      if not exists (select * from contas where numero = @num)
         begin rollback; return -3; end
      update contas set saldo = saldo + @valor where numero = @num
  commit
  return 0
end

create proc transferir @num1 int, @num2 int, @valor real
as
begin
   declare @r int
   begin tran
     exec @r = debitar @num1,@valor
     if @r <> 0 begin rollback; return @r; end
     exec @r = creditar @num2,@valor
     if @r <> 0 begin rollback; return @r; end
  commit
  return 0;
end
```

Pretende-se que os procedimentos armazenados debitar e creditar possam ser autónomos do ponto de vista transacional, mas estejam preparados para serem chamados no âmbito de uma transação já existente. Porém, quando se pretendeu realizar uma transferência entre duas contas existentes, mas não havendo saldo suficiente na primeira conta, obteve.se o seguinte erro:

```
Msg 266, Level 16, State 2, Procedure debitar, …
Transaction count after EXECUTE indicates a mismatching number of BEGIN and COMMIT statements. …
```

a) Altere o código de forma a eliminar este erro, mas mantendo os requisitos antes enunciados.
b) Tratando-se de uma implementação do modelo de transações hierárquicas, indique como é garantida nesta implementação a regra de visibilidade desse modelo.

---

5) Usando **System.Transactions.TransactionScope**, implemente o código c# equivalente ao código do exercício 4. Admita a existência de um *mapper* que implementa as operações CRUD sobre a tabela contas.

Cotação:

alínea	1.a	1.b	1.c	2.a	2.b	2.c	3.a	3.b	3.c	4.a	4.b	5	**Total**
cotação	2	2	2	2	2	2	1	1	1	2	1	2	20