ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Sistemas de Informação II

Turma LI52D | Inverno 2017/2018

Relatório 1ºTrabalho

Sistema de Informação II

Filipe Fé 42141 | Rui Lima 42200

Docente: Engenheiro Nuno Datia

Descrição

Neste trabalho era pedido a realização de um modelo EA que desse suporte para a criação de um sistema de informação da empresa Glampinho que permitisse gerir os seus parques de campismo de luxo, respeitando as entidades apresentadas no enunciado juntamente com as restrições de integridade também nele referidas. Foi também pedido a criação de código T-SQL que ilustrasse as ‘’queries’’ pedidas em cada umas das alíneas e por fim testes que comprovassem o bom funcionamento das mesmas.

Modelo Relacional

Simples:

- ParqueCampismo(nome [nvarchar(30)], morada[nvarchar(50)], estrelas[tinyint in(1, 2, 3, 4, 5)], email[nvarchar(30)])
- Extra(id[int], descrição[nvarchar(30)], preçoDia[int], associado[varchar(10) in('alojamento', 'pessoa')])
    - O atributo associado indica se o extra é facturado por alojamento ou pessoa
- Hóspede(NIF[int], nome[nvarchar(30)], morada[nvarchar(50)], email[nvarchar(30)], númeroIdentificação[int])
- Estada(id[int], dataInício[DateTime2], dataFim[DateTime2], idFactura[int], ano[int])
    - O par idFactura-ano é chave estrangeira para Factura
    - data de início tem de ser menor que a data de fim
- Factura(id [int], ano[int], nomeHóspede[nvarchar(30)], NIFHóspede[int], preçoTotal[int])
    - NIFHóspede é uma chave estrangeira para Hóspede.NIF
    - nomeHóspede tem de ser o nome do hóspede associado ao NIFHóspede.
    - preçoTotal indica o custo total da estada para a qual foi emitida a factura

Fracas:

- Alojamento(nomeParque[nvarchar(30)], localização[nvarchar(30)], nome[nvarchar(30)], descrição[nvarchar(30)], preçoBase[int], númeroMáximoPessoas[tinyInt], tipoAlojamento[varchar(8) in('bungalow' , 'tenda')])
    - nomeParque é chave estrangeira para ParqueCampismo.nome.
    - nome é chave candidata
- Bungalow(nomeParque[nvarchar(30)], localização[nvarchar(30)], tipologia[char(2)])
    - O par nomeParque-localização é chave estrangeira para Alojamento
    - Tipologia vem na forma 'Tx', estando x no intervalo [0,3]
- Tenda(nomeParque[nvarchar(30)], localização[nvarchar(30)], área[int])
    - O par nomeParque-localização é chave estrangeira para Alojamento
- Actividades(nomeParque[nvarchar(30)], númeroSequencial[int], ano[int], nome[nvarchar(30)], descrição[nvarchar(30)], lotaçãoMáxima[int], preçoParticipante[int], dataRealização[DateTime2])
    - nomeParque é chave estrangeira para ParqueCampismo.nome
- Telefones(nomeParque[nvarchar(30)], telefone[int])
    - nomeParque é chave estrangeira para ParqueCampismo.nome
- Item(idFactura[int], linha[int], quantidade[int], preço[int], descrição[nvarchar(30)], tipo[varchar(10) in ('actividade', 'alojamento', 'extra')])
    - idFactura é chave estrangeira para Factura.id
    - quantidade indica quanto vezes foi esse item usufruído (número de pessoas * número de dias)
    - preço indica o total a pagar para esse item, já calculado tendo em conta a quantidade

- descrição é a descrição correspondente a esse item

Associações:

- Paga(nomeParque[nvarchar(30)], númeroSequencial[int], ano[int], NIF[int], preçoParticipante[int])
  - O conjunto nomeParque-númeroSequencial-ano é chave estrangeira para Actividades(nomeParque, númeroSequencial, ano)
  - NIF é chave estrangeira para Hóspede.
- HóspedeEstada(NIF[int], id[int], hóspede[true,false])
  - NIF é chave estrangeira para Hóspede.NIF
  - id é chave estrangeira para Estada.id
  - hóspede identifica se o hóspede associado à estada é o responsável ou não.
- EstadaExtra(estadaId[int], extraId[int], preçoDia[int])
  - estadaId é chave estrangeira para estada.id
  - extraId é chave estrangeira para extra.id
  - preçoDia indica o preço do extra no momento da criação da estada
- AlojamentoEstada(nomeParque[nvarchar(30), localização[nvarchar(30)], id[int], preçoBase[int])
  - O par nomeParque-localização é chave estrangeira para Alojamento
  - id é chave estrangeira para Estada.id
  - preçoBase indica qual o custo do alojamento aquando da criação da estada
- AlojamentoExtra(nomeParque[nvarchar(30)], localização[nvarchar(30)], id[int])
  - O par nomeParque-localização é chave estrangeira para Alojamento
  - id é chave estrangeira para Extra.id

Restrições de integridade

- ***Alojamento é caracterizado por um nome único e por uma localização representada por um conjunto caracteres alfanuméricos únicos dentro de cada parque*.**

  Inicialmente foi definido como chave primária o conjunto ***nomeParque,nome,localização***. Para respeitar a 3FN , foi retirado o atributo nome da chave primária de Alojamento, visto que basta o par ***localização,nome*** para identificar um Alojamento dentro do Parque de Campismo. Dessa forma o atributo nome passou a ser chave candidata pois, por ser único, também ele consegue identificar o alojamento dentro do parque.

- ***Qualquer alteração de preços de alojamento e de extras posterior a uma reserva ou início de estada não irá alterar o valor a pagar pelos hóspedes.***

  A nossa solução para esta restrição foi atribuir às relações entre entidades que tenham preços associados e a estada um atributo preço que especifica o preço definido na altura da reserva.

- ***Um hóspede, quando criado, tem de estar associado a uma estada num determinado Alojamento do parque.***

  É garantido que no momento a seguir à inserção do hóspede, este é associado a uma estada já existente se não for o responsável ou caso seja responsável, é criada a estada na altura.

- ***Uma estada no parque fica associada a um hóspede responsável, e tem de ter associados um ou mais alojamentos e uma ou mais pessoas.***

- ***Os mesmos hóspedes podem ficar alojados diversas vezes no mesmo alojamento, desde que em períodos diferentes.***
- ***Os extras para cada estada são definidos no momento do registo no sistema da estada.*** Desta forma, não é possível associar um extra novo a meio de uma estada. Por outro lado facilita a emissão da fatura para qualquer estada que tenha extras associados.
- ***A data de fim de uma estada é especificada no início de uma reserva.***
- ***Consequentemente, a fatura é emitida depois de criação da estada ser feita, tendo a ela associada todos os extras, atividades e alojamento usados na estada.***
- ***Uma fatura não pode ser apagada do sistema de modo a manter sempre um registo das contas da empresa***
- ***Aquando da eliminação da base de dados de um hóspede responsável é eliminada também a(s) estada(s) a que era responsável.***

  Assim, são eliminados todos os dados referentes a essa estada, que incluem, a associação AlojamentoEstada, que indica qual ou quais os alojamentos daquela estada, a associação ExtraEstada, que indica quais os extras que foram usufruídos naquela estada, e a associação HóspedeEstada, que indica os hóspedes daquela estada. No momento da eliminação da associação HóspedeEstada, caso o hóspede apenas esteja inscrito na base de dados devido a essa estada, não participou noutra estada, também ele é eliminado da base de dados.

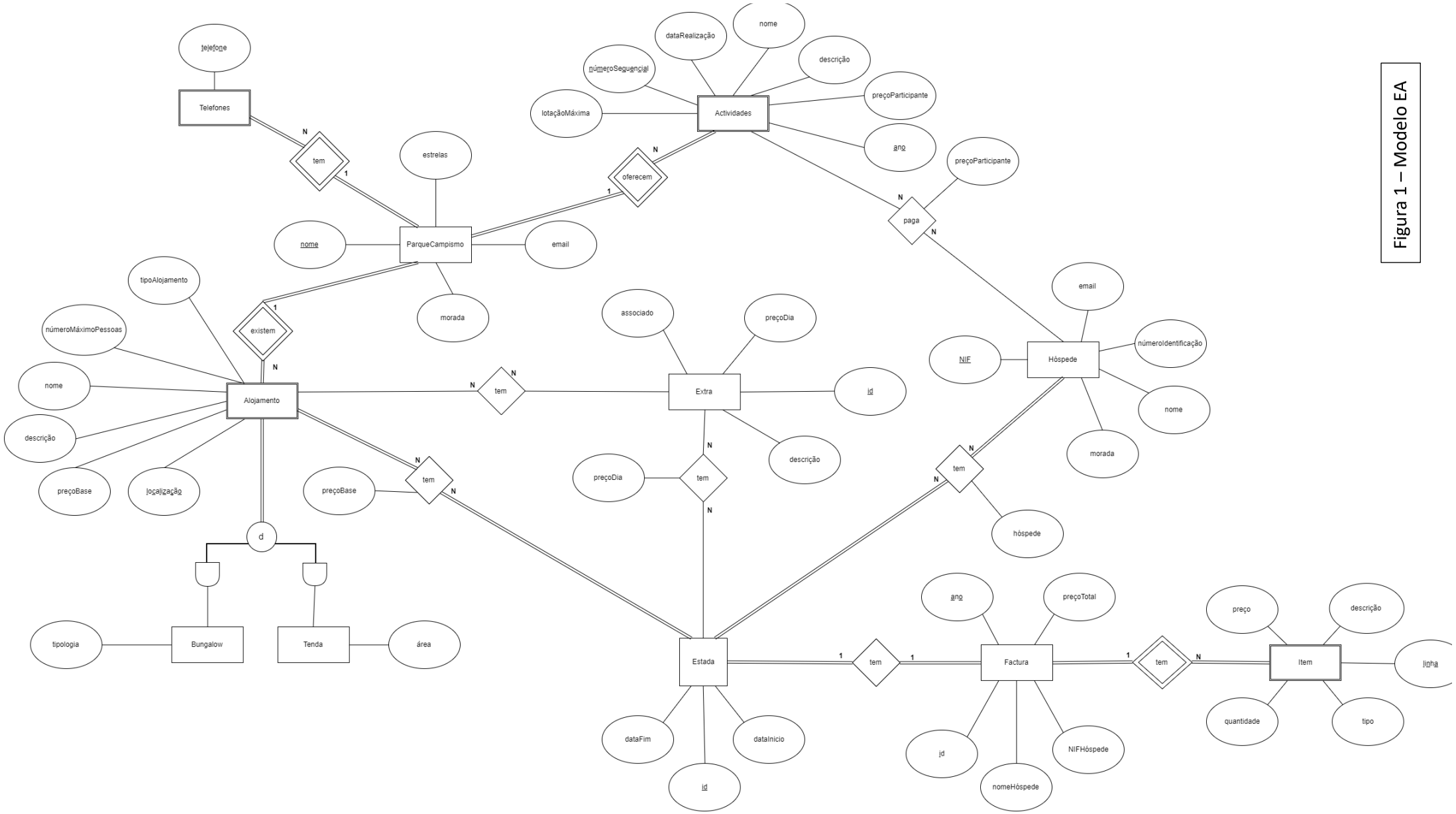


Figura 1 – Modelo EA

Resposta às alíneas do trabalho

a) ***Criação do modelo físico.***

Foi realizado um script com a criação de todas as entidades necessárias, inclusive as associações entre elas, respeitando as restrições de integridade que são possíveis implementar em código SQL.

b) ***Remover o modelo físico.***

Foi realizado um script com os drops de todas as tabelas do nosso modelo de dados, com o cuidado de ao apagar uma tabela, que não existam ainda outras com chaves estrangeiras para a mesma

c) ***Inserir, remover e atualizar informação de um hóspede.***

Não foi necessário a criação de um procedimento armazenado para inserir e atualizar a informação de um hóspede, visto que trata apenas de uma instrução de INSERT/UPDATE na respetiva entidade.

```
/******************************** INSERT ************************************************************/
INSERT INTO dbo.Hóspede(NIF, nome, morada, email, númeroIdentificação)
        VALUES (112233445, 'José', 'Rua 1', 'jose@gmail.com', 11223344),
               (566778899, 'Maria', 'Rua 2', 'maria@gmail.com', 55667788)

/******************************** UPDATE ************************************************************/

UPDATE dbo.Hóspede SET morada = 'Rua Teste' WHERE NIF = 112233445
```

Para apagar um hóspede é necessário ter o cuidado de verificar se o hóspede em causa é o responsável pela estada. Se for, é necessário eliminar a informação relativa a esse hóspede nas tabelas que representam associações entre ele e a estada. Consequentemente a estada é também ela apagada do sistema para garantir o cumprimento da restrição de integridade que afirma que qualquer estada tem de ter um hóspede responsável associado. Dessa forma, todos os hóspedes que estavam alojados na mesma estada do responsável têm de ser apagados do sistema. Para isso é chamado o procedimento armazenado dbo.eliminaHóspedesAssociados, o qual depois da eliminação do tuplo referente a estada a ser eliminada, verifica se o hóspede removido está ou esteve inscrito em alguma outra estada, em caso negativo é também ele eliminado do sistema. Este procedimento tem o nível de isolamento REPEATABLE READ de modo a evitar as anomalias dirty read e nonrepeatable read

O nível de isolamento desta transação foi definido como REPEATABLE READ de forma a evitar lost updates.

```
CREATE PROCEDURE dbo.deleteHospede @NIFHospede INT AS
BEGIN TRY
    BEGIN TRANSACTION SET TRANSACTION ISOLATION LEVEL REPEATABLE READ -- para evitar lost updates
                DECLARE @hóspede VARCHAR(5)
                DECLARE @idEstada INT
                DECLARE eliminaEstadaInfo CURSOR FOR SELECT id, hóspede FROM dbo.HóspedeEstada WHERE NIF = @NIFHospede

                OPEN eliminaEstadaInfo
                FETCH FROM eliminaEstadaInfo INTO @idEstada, @hóspede

                WHILE @@FETCH_STATUS = 0
                        BEGIN
                                if @hóspede = 'true'    -- significa que deve ser elimina a estada e todas as entidades a
                                        BEGIN
                                                DELETE FROM dbo.AlojamentoEstada WHERE id = @idEstada
                                                DELETE FROM dbo.EstadaExtra WHERE estadaId = @idEstada
                                                DELETE FROM dbo.HóspedeEstada WHERE NIF = @NIFHospede

                                                EXEC dbo.eliminaHóspedesAssociados @idEstada    -- verifica se a estada e

                                                DELETE FROM dbo.Estada WHERE id = @idEstada
                                                DELETE FROM dbo.Hóspede WHERE NIF = @NIFHospede
                                        END
                                ELSE
                                        BEGIN
                                                DELETE FROM dbo.HóspedeEstada WHERE NIF = @NIFHospede
                                                DELETE FROM dbo.Hóspede WHERE NIF = @NIFHospede
                                        END

                                FETCH FROM eliminaEstadaInfo INTO @idEstada, @hóspede
                        END
    COMMIT
END TRY
BEGIN CATCH
        IF @@TRANCOUNT !=0
                ROLLBACK;
        THROW
END CATCH
```

```
IF EXISTS(SELECT 1 FROM sys.objects WHERE type_desc = 'SQL_STORED_PROCEDURE' AND name = 'eliminaHóspedesAssociados')
    DROP PROCEDURE dbo.eliminaHóspedesAssociados;
GO
CREATE PROCEDURE dbo.eliminaHóspedesAssociados @idEstada INT AS
SET NOCOUNT ON
BEGIN TRY
    BEGIN TRANSACTION SET TRANSACTION ISOLATION LEVEL REPEATABLE READ   -- evita as anomalias dirty read e nonrepeatable read
        DECLARE @NIFHóspede INT
        DECLARE @count INT

        DECLARE hóspedesAssociados CURSOR FOR SELECT NIF FROM dbo.HóspedeEstada WHERE id = @idEstada

        OPEN hóspedesAssociados
        FETCH NEXT FROM hóspedesAssociados INTO @NIFHóspede

        WHILE @@FETCH_STATUS = 0
        BEGIN
            DELETE FROM dbo.HóspedeEstada WHERE NIF = @NIFHóspede AND id = @idEstada

            SELECT @count = COUNT(id) FROM dbo.HóspedeEstada WHERE NIF = @NIFHóspede

            IF @count = 0
            BEGIN
                DELETE FROM dbo.Hóspede WHERE NIF = @NIFHóspede
            END

            FETCH NEXT FROM hóspedesAssociados INTO @NIFHóspede
        END

        CLOSE hóspedesAssociados
        DEALLOCATE hóspedesAssociados
    COMMIT
END TRY
BEGIN CATCH
    IF @@TRANCOUNT !=0
        ROLLBACK;
    THROW
END CATCH
```

d) Inserir, remover e atualizar informação de um alojamento num parque.

Não foi necessário a criação de um procedimento armazenado para atualizar a informação de um alojamento, visto que se trata apenas de uma instrução de UPDATE na respetiva entidade.

```
/******************************** UPDATE ************************************************************/

UPDATE dbo.Alojamento SET preçoBase = 80 WHERE nome = 'Parque 1' and localização = 'Lote 1'
```

Para a inserção de um alojamento há que ter atenção os seguintes aspetos:

- Verificar qual o tipo de Alojamento a ser criado;
- Inserir na tabela correspondente ao tipo de Alojamento referido (Tenda ou Bungalow);
- Adicionar o atributo área ou tipologia, consoante o tipo;
- Criar a entidade Alojamento respetiva.

Por essa razão foram criados dois procedimentos armazenados com o objetivo de distinguir o tipo de alojamento e dessa forma inserir na entidade respetiva. Ao invés de escolher um único procedimento armazenado que tratasse de todos dos aspetos referidos em cima e que recebesse como parâmetro os dois atributos área e tipologia, sendo que há partida um deles viria com o valor NULL, optámos por subdividi-lo em dois, para tornar o código mais legível e funcional.

Foi definido o nível de isolamento da transação para READ COMMITED porque uma vez que se tratam de transações apenas com inserts, os quais já incluem um lock, foi colocado este nível de isolamento pois trata-se do nível de isolamento por omissão e assim repõe-se os isolamentos da base de dados independentemente do nível de isolamento anterior

```
/****************************** INSERT Bungalow ************************************************/
GO
CREATE PROCEDURE dbo.InsertAlojamentoBungalow @nomeParque NVARCHAR(30), @nome NVARCHAR(30), @localização NVARCHAR(30),
                                                                                    @descrição NVARCHAR(30), @preçoBase INT, @númeroM
BEGIN TRY
        BEGIN TRANSACTION SET TRANSACTION ISOLATION LEVEL READ COMMITTED
                INSERT INTO dbo.Alojamento(nomeParque, nome, localização, descrição, preçoBase, númeroMáximoPessoas, tipoAlojamento)
                        VALUES(@nomeParque, @nome, @localização, @descrição, @preçoBase, @númeroMáximoPessoas, 'bungalow')

                INSERT INTO dbo.Bungalow(nomeParque, localização, tipologia)
                        VALUES(@nomeParque, @localização, @tipologia)
        COMMIT
END TRY
BEGIN CATCH
        IF @@TRANCOUNT !=0
                ROLLBACK;
        THROW
END CATCH

/****************************** INSERT Tenda ***************************************************/
GO
CREATE PROCEDURE dbo.InsertAlojamentoTenda @nomeParque NVARCHAR(30), @nome NVARCHAR(30), @localização NVARCHAR(30),
                                                                                @descrição NVARCHAR(30), @preçoBase INT, @númeroMáximoPe
BEGIN TRY
        BEGIN TRANSACTION SET TRANSACTION ISOLATION LEVEL READ COMMITTED
                INSERT INTO dbo.Alojamento(nomeParque, nome, localização, descrição, preçoBase, númeroMáximoPessoas, tipoAlojamento)
                        VALUES(@nomeParque, @nome, @localização, @descrição, @preçoBase, @númeroMáximoPessoas, 'tenda')

                INSERT INTO dbo.Tenda(nomeParque, localização, área)
                        VALUES(@nomeParque, @localização, @área)
                COMMIT
END TRY
BEGIN CATCH
        IF @@TRANCOUNT !=0
                ROLLBACK;
        THROW
```

Para apagar um Alojamento é necessário apagá-lo primeiro das tabelas que representam as associações entre outras entidades e o alojamento com o nome e nome do Parque passados como parâmetro, e caso o alojamento seja o único de uma dada estada, também esta, bem como, os elementos da associação HóspedeEstada, referentes à estada a eliminar, e, por conseguinte, os hóspedes que apenas estiveram nessa estada, no fim apagar da tabela Alojamento. Para isso foi criado o procedimento *deleteAlojamento*, com um nível de isolamento READ COMMITED, uma vez que as instruções a executar são deletes, que tal como os insert, têm um lock próprio, sendo por isso reposto o nível de isolamento por omissão da base de dados. Para eliminar os hóspedes é usado o procedimento *eliminaHóspedesAssociados*, tal como na alínea C.

```
/********************************* DELETE ALOJAMENTO ***************************************************************/
GO
IF EXISTS(SELECT 1 FROM sys.objects WHERE type_desc = 'SQL_STORED_PROCEDURE' AND name = 'deleteAlojamento')
    DROP PROCEDURE dbo.deleteAlojamento;

GO
CREATE PROCEDURE dbo.deleteAlojamento @localização NVARCHAR(30), @nomeParque NVARCHAR(30) AS
SET NOCOUNT ON
BEGIN TRY
    BEGIN TRANSACTION SET TRANSACTION ISOLATION LEVEL READ COMMITTED
        DECLARE @idEstada INT
        DECLARE eliminaAlojamentoInfo CURSOR FOR SELECT id FROM dbo.AlojamentoEstada WHERE nomeParque = @nomeParque AND localização = @localização

        OPEN eliminaAlojamentoInfo
        FETCH NEXT FROM eliminaAlojamentoInfo INTO @idEstada

        WHILE @@FETCH_STATUS = 0
        BEGIN
            IF (SELECT COUNT(id) FROM dbo.AlojamentoEstada WHERE id = @idEstada) = 1
                BEGIN
                    EXEC dbo.eliminaHóspedesAssociados @idEstada
                    DELETE FROM dbo.AlojamentoEstada WHERE id = @idEstada
                    DELETE FROM dbo.EstadaExtra WHERE estadaId = @idEstada
                    DELETE FROM dbo.Estada WHERE id = @idEstada

                END
                ELSE
                    DELETE FROM dbo.AlojamentoEstada WHERE nomeParque = @nomeParque AND localização = @localização
                FETCH NEXT FROM eliminaAlojamentoInfo INTO @idEstada
        END

        CLOSE eliminaAlojamentoInfo
        DEALLOCATE eliminaAlojamentoInfo

        DELETE FROM dbo.AlojamentoExtra WHERE nomeParque = @nomeParque AND localização = @localização
        DELETE FROM dbo.Alojamento WHERE nomeParque = @nomeParque AND localização = @localização
    COMMIT
END TRY
BEGIN CATCH
    IF @@TRANCOUNT !=0
        ROLLBACK;
    THROW
END CATCH
```

e) Inserir, remover e atualizar informação de um extra de alojamento.

A inserção e atualização da informação de um extra, quer seja do tipo pessoal ou do tipo de alojamento é realizado da mesma forma: foi criado na entidade Extra um atributo "associado" que permite distinguir precisamente qual o tipo de extra a que nos estamos a referir. Posto isto, para inserir/atualizar é apenas necessário realizar uma instrução INSERT/UPDATE com o atributo associado especificado e os restantes que serão inseridos/atualizados.

```
/********************************* INSERT ***************************************************************/

INSERT INTO dbo.Extra(id, descrição, preçoDia, associado)
        VALUES(2, 'descricao', 10, 'alojamento')

/********************************* UPDATE ***************************************************************/

UPDATE dbo.Extra SET preçoDia = preçoDia - 2 WHERE id = 2
```

Para remover é necessário primeiramente remover das entidades que se associam com Extra (Alojamento e Estada) os tuplos que tenham o mesmo id de Extra passado como parâmetro no procedimento armazenado de delete, e por fim apagar da tabela Extra. Isto acontece caso o extra que corresponde ao id passado seja do tipo alojamento. Para isso foi criado um procedimento, que executa os deletes referidos e tem como nível de isolamento READ COMMITED, uma vez que são executadas instruções de delete que por si só já possui um lock, sendo apenas necessário repor o nível de isolamento por omissão do SQL Server, que poderia ter sido alterado por um outro procedimento.

```
/********************************** DELETE ***********************************************************/
GO
IF EXISTS(SELECT 1 FROM sys.objects WHERE type_desc = 'SQL_STORED_PROCEDURE' AND name = 'deleteExtra'
    DROP PROCEDURE dbo.deleteExtra;
GO
CREATE PROCEDURE dbo.deleteExtra @id INT AS
SET NOCOUNT ON
BEGIN TRY
    BEGIN TRANSACTION SET TRANSACTION ISOLATION LEVEL READ COMMITTED
        IF NOT EXISTS(SELECT 1 FROM dbo.Extra WHERE id = @id AND associado = 'alojamento')
            THROW 51000, 'O extra tem de ser de alojamento', 1
        DELETE FROM dbo.AlojamentoExtra WHERE id=@id
        DELETE FROM dbo.EstadaExtra WHERE estadaId=@id
        DELETE FROM dbo.Extra WHERE id=@id
    COMMIT
END TRY
BEGIN CATCH
    IF @@TRANCOUNT !=0
        ROLLBACK;
    THROW
END CATCH

/********************************** TESTE ************************************************************/
```

f) ***Inserir, remover e atualizar informação de um extra pessoal***.

Esta alínea é muito semelhante à alínea anterior, sendo a única diferença, referente ao tipo de extra a que se refere, sendo neste caso extras do tipo pessoal

```
13  /********************************** INSERT ***********************************************************/
14
15  INSERT INTO dbo.Extra(id, descrição, preçoDia, associado)
16      VALUES(3, 'descricao', 12, 'pessoa')
17
18  /********************************** UPDATE ***********************************************************/
19
20  UPDATE dbo.Extra SET preçoDia = preçoDia - 2 WHERE id = 2
21
22  /********************************** DELETE ***********************************************************/
23  GO
24  IF EXISTS(SELECT 1 FROM sys.objects WHERE type_desc = 'SQL_STORED_PROCEDURE' AND name = 'deleteExtraPessoa')
25      DROP PROCEDURE dbo.deleteExtraPessoa;
26  GO
27  CREATE PROCEDURE dbo.deleteExtraPessoa @id INT AS
28  SET NOCOUNT ON
29  BEGIN TRY
30      BEGIN TRANSACTION SET TRANSACTION ISOLATION LEVEL READ COMMITTED
31          IF NOT EXISTS(SELECT 1 FROM dbo.Extra WHERE id = @id AND associado = 'pessoa')
32              THROW 51000, 'O extra tem de ser de pessoa', 1
33          DELETE FROM dbo.AlojamentoExtra WHERE id=@id
34          DELETE FROM dbo.EstadaExtra WHERE estadaId=@id
35          DELETE FROM dbo.Extra WHERE id=@id
36      COMMIT
37  END TRY
38  BEGIN CATCH
39      IF @@TRANCOUNT !=0
40          ROLLBACK;
41      THROW
42  END CATCH
43
```

g) *Inserir, remover e atualizar informação de uma atividade.*

Não foi necessário a criação de um procedimento armazenado para inserir e atualizar a informação de um hóspede, visto que trata apenas de uma instrução de INSERT/UPDATE na respetiva entidade.

```
/************************************** INSERT ************************************************/

INSERT INTO dbo.Actividades(nomeParque, númeroSequencial, nome, descrição, lotaçãoMáxima, preçoParticipante, dataRealização)
        VALUES('Glampinho', 1, 'FUT7', 'Jogo de futebol 7vs7', '14', 3, '03-15-17 10:30')

/************************************** UPDATE ************************************************/

UPDATE dbo.Actividades SET preçoParticipante = preçoParticipante - 2 WHERE nomeParque = 'Glampinho' AND númeroSequencial = 1
```

Para remover a informação da atividade é preciso remover primeiro da tabela Paga e só posteriormente remover da tabela Atividade. Sendo estes deletes feitos num procedimento armazenado com o nível de isolamento READ COMMITED, pois os deletes já têm o seu próprio lock de proteção e assim é reposto o nível de isolamento por omissão do SQL Server.

```
GO
CREATE PROCEDURE dbo.deleteAtividades @nomeParque NVARCHAR(30), @númeroSequencial INT AS
BEGIN TRY
        BEGIN TRANSACTION SET TRANSACTION ISOLATION LEVEL READ COMMITTED
                DELETE FROM dbo.Paga WHERE nomeParque = @nomeParque and númeroSequencial = @númeroSequencial
                DELETE FROM dbo.Actividades WHERE nomeParque = @nomeParque and númeroSequencial = @númeroSequencial
        COMMIT
END TRY
BEGIN CATCH
        IF @@TRANCOUNT !=0
                ROLLBACK;
        THROW
END CATCH
```

h) *Criar uma estada para um dado período de tempo. Este processamento deve ser dividido nos seguintes sub-processamentos.*

```
GO
CREATE PROCEDURE dbo.createEstadaInTime @NIFResponsável INT, @NIFHóspede INT, @tempoEstada INT, @tipoAlojamento VARCHAR(8), @lotação TI
        BEGIN TRY
                BEGIN TRANSACTION
                        DECLARE @id INT
                        EXEC dbo.createEstada @NIFResponsável, @tempoEstada, @id OUTPUT

                        EXEC dbo.addAlojamento @tipoAlojamento, @lotação, @id

                        EXEC dbo.addHóspede @NIFHóspede, @id

                        EXEC dbo.addExtraToAlojamento @idExtraAlojamento, @id

                        EXEC dbo.addExtraToEstada @idExtraPessoal, @id
                COMMIT
        END TRY
        BEGIN CATCH
                IF @@TRANCOUNT !=0
                        ROLLBACK;
                THROW
        END CATCH
```

1. Criar uma estada, recebendo como parâmetro o NIF do responsável e a duração da estada. É inserido na tabela estada a data atual como data de inicio da estada, e soma-se a duração à data atual de forma a definir a data de fim. É verificado se já existe algum hóspede responsável pela estada em causa.
2. Adicionar um alojamento à estada criada anteriormente, desde que obedeça às condições passadas como parâmetro, que neste caso é a lotação. Verificar se existem alojamento disponíveis para poder ser feita a reserva. Se sim, então é adicionado na altura os Extras desejados para aquela estada e é permitido avançar para o procedimento seguinte.
3. É adicionado um hóspede à estada criada.
4. Adicionados os extras para aquele tipo de Alojamento com o preço da altura.
5. Por fim é adicionado um extra pessoal e a transação é feita com sucesso.

RESULTADO ESPERADO: (nota: já existiam 5 estadas na base de dados, daí o id gerado ser o 6)

```
EXEC dbo.createEstadaInTime 112233445, 566778899, 5, 'tenda', 4, 2, 1

SELECT * FROM Estada WHERE id = 6
SELECT * FROM HóspedeEstada WHERE id=6
SELECT * FROM AlojamentoEstada WHERE id=6
SELECT * FROM EstadaExtra WHERE estadaId=6
```

sults | Messages

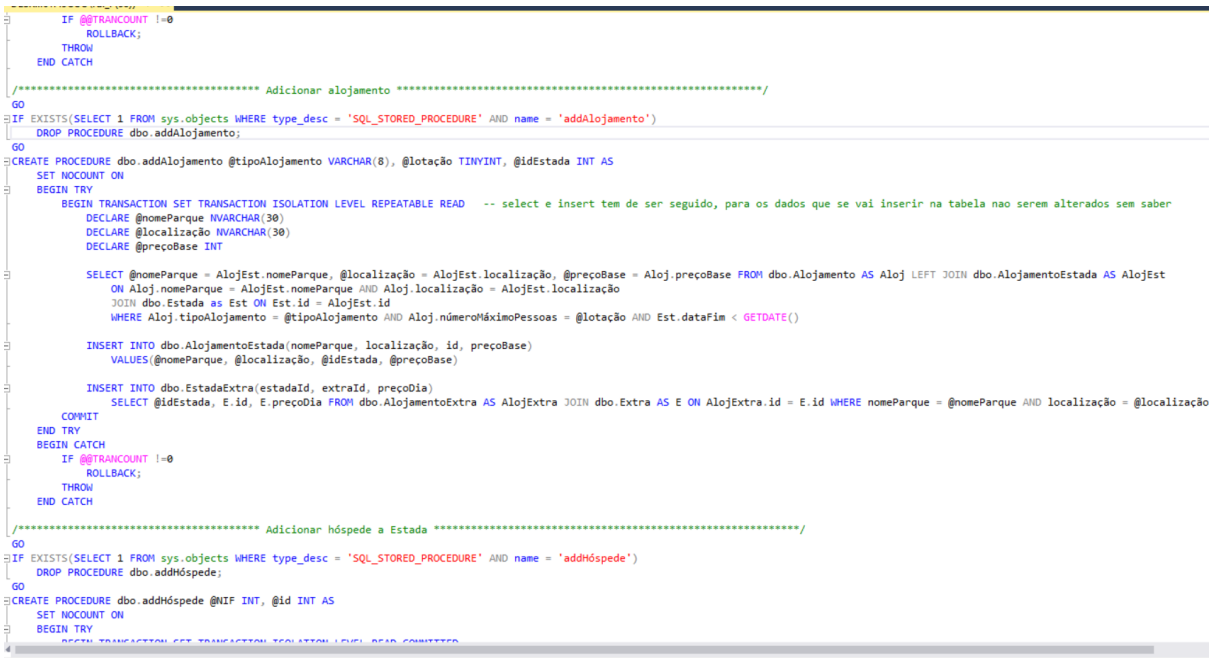
id	dataInício	dataFim	idFact...	ano
6	2017-11-18 14:24:57.0300000	2017-11-23 14:24:57.0300000	NULL	NULL

NIF	id	hóspe...
112233445	6	true
566778899	6	false

nomeParq...	localizaç...	id	preçoBa...
Glampinho	Rua 1	6	NULL

estad...	extraId	preço...
6	2	15

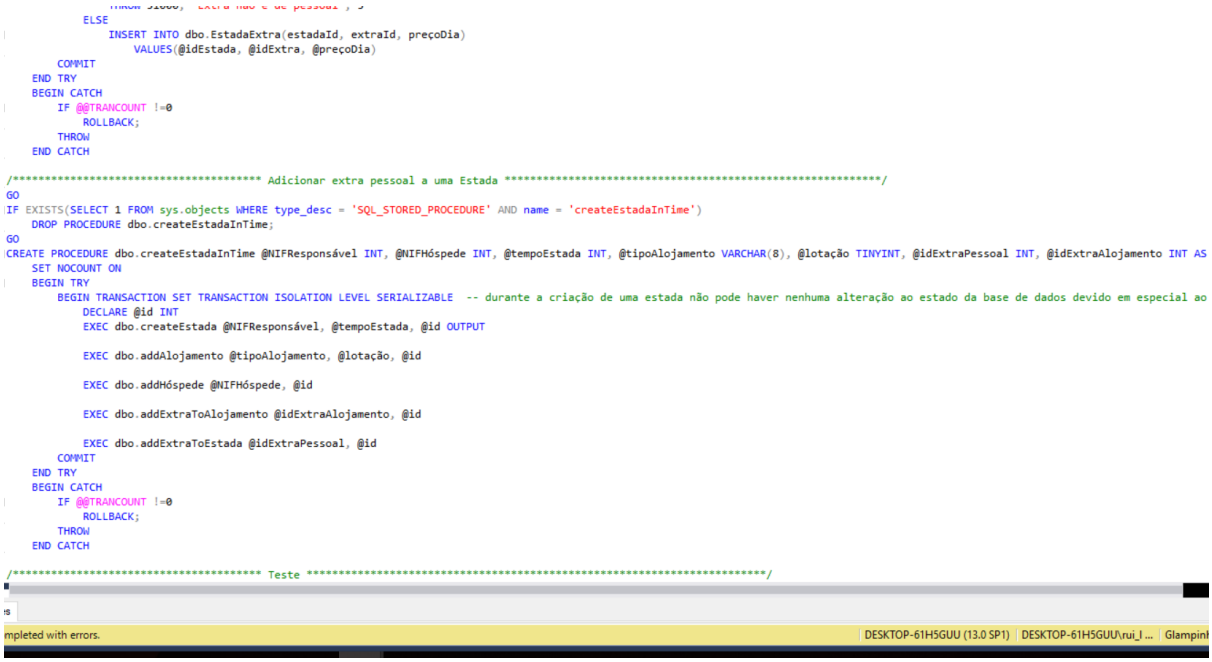
```
/************************************ Criar uma estada ********************************************************************/
GO
IF EXISTS(SELECT 1 FROM sys.objects WHERE type_desc = 'SQL_STORED_PROCEDURE' AND name = 'createEstada')
    DROP PROCEDURE dbo.createEstada;
GO
CREATE PROCEDURE dbo.createEstada @NIFResponsável INT, @tempoEstada INT, @idNumber INT OUTPUT AS -- em minutos
    SET NOCOUNT ON
    BEGIN TRY
        BEGIN TRANSACTION SET TRANSACTION ISOLATION LEVEL READ COMMITTED
            DECLARE @date DATETIME2

            SELECT @date = GETDATE()

            SELECT @idNumber = MAX(id) + 1 FROM dbo.Estada

            INSERT INTO dbo.Estada(id, dataInício, dataFim)
                VALUES(@idNumber, @date, DATEADD(DAY, @tempoEstada, @date))

            INSERT INTO dbo.HóspedeEstada(NIF, id, hóspede)
                VALUES(@NifResponsável, @idNumber, 'true')

        COMMIT
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT !=0
            ROLLBACK;
        THROW
    END CATCH
```



```
        IF @@TRANCOUNT !=0
            ROLLBACK;
        THROW
    END CATCH

/************************************ Adicionar alojamento ********************************************************/
GO
IF EXISTS(SELECT 1 FROM sys.objects WHERE type_desc = 'SQL_STORED_PROCEDURE' AND name = 'addAlojamento')
    DROP PROCEDURE dbo.addAlojamento;
GO
CREATE PROCEDURE dbo.addAlojamento @tipoAlojamento VARCHAR(8), @lotação TINYINT, @idEstada INT AS
    SET NOCOUNT ON
    BEGIN TRY
        BEGIN TRANSACTION SET TRANSACTION ISOLATION LEVEL REPEATABLE READ   -- select e insert tem de ser seguido, para os dados que se vai inserir na tabela nao serem alterados sem saber
            DECLARE @nomeParque NVARCHAR(30)
            DECLARE @localização NVARCHAR(30)
            DECLARE @preçoBase INT

            SELECT @nomeParque = AlojEst.nomeParque, @localização = AlojEst.localização, @preçoBase = Aloj.preçoBase FROM dbo.Alojamento AS Aloj LEFT JOIN dbo.AlojamentoEstada AS AlojEst
                ON Aloj.nomeParque = AlojEst.nomeParque AND Aloj.localização = AlojEst.localização
                JOIN dbo.Estada as Est ON Est.id = AlojEst.id
                WHERE Aloj.tipoAlojamento = @tipoAlojamento AND Aloj.númeroMáximoPessoas = @lotação AND Est.dataFim < GETDATE()

            INSERT INTO dbo.AlojamentoEstada(nomeParque, localização, id, preçoBase)
                VALUES(@nomeParque, @localização, @idEstada, @preçoBase)

            INSERT INTO dbo.EstadaExtra(estadaId, extraId, preçoDia)
                SELECT @idEstada, E.id, E.preçoDia FROM dbo.AlojamentoExtra AS AlojExtra JOIN dbo.Extra AS E ON AlojExtra.id = E.id WHERE nomeParque = @nomeParque AND localização = @localização
        COMMIT
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT !=0
            ROLLBACK;
        THROW
    END CATCH

/************************************ Adicionar hóspede a Estada ********************************************************/
GO
IF EXISTS(SELECT 1 FROM sys.objects WHERE type_desc = 'SQL_STORED_PROCEDURE' AND name = 'addHóspede')
    DROP PROCEDURE dbo.addHóspede;
GO
CREATE PROCEDURE dbo.addHóspede @NIF INT, @id INT AS
    SET NOCOUNT ON
    BEGIN TRY
```

```
        THROW
    END CATCH

/************************************ Adicionar hóspede a Estada ********************************
GO
IF EXISTS(SELECT 1 FROM sys.objects WHERE type_desc = 'SQL_STORED_PROCEDURE' AND name = 'addHóspede')
    DROP PROCEDURE dbo.addHóspede;
GO
CREATE PROCEDURE dbo.addHóspede @NIF INT, @id INT AS
    SET NOCOUNT ON
    BEGIN TRY
        BEGIN TRANSACTION SET TRANSACTION ISOLATION LEVEL READ COMMITTED
            INSERT INTO dbo.HóspedeEstada(NIF, id, hóspede)
                VALUES(@NIF, @id, 'false')
        COMMIT
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT !=0
            ROLLBACK;
        THROW
    END CATCH

/************************************ Adicionar extra a um alojamento de uma Estada **************
GO
```

```sql
/************************************ Adicionar extra a um alojamento de uma Estada ****************************************
GO
IF EXISTS(SELECT 1 FROM sys.objects WHERE type_desc = 'SQL_STORED_PROCEDURE' AND name = 'addExtraToAlojamento')
    DROP PROCEDURE dbo.addExtraToAlojamento;
GO
CREATE PROCEDURE dbo.addExtraToAlojamento @idExtra INT, @idEstada INT AS
    SET NOCOUNT ON
    BEGIN TRY
        BEGIN TRANSACTION SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
            DECLARE @nomeParque NVARCHAR(30)
            DECLARE @localização NVARCHAR(30)
            DECLARE @associado VARCHAR(10)

            SELECT @associado = associado FROM dbo.Extra WHERE id = @idExtra

            IF(@associado <> 'alojamento')
                THROW 51000, 'Extra não é de pessoal', 5
            ELSE
                BEGIN
                    SELECT @nomeParque = nomeParque, @localização = localização FROM dbo.AlojamentoEstada WHERE id = @idEstada

                    INSERT INTO dbo.AlojamentoExtra(nomeParque, localização, id)
                        VALUES(@nomeParque, @localização, @idExtra)
                END
        COMMIT
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT !=0
            ROLLBACK;
        THROW
    END CATCH



/************************************ Adicionar extra pessoal a uma Estada ************************************
GO
IF EXISTS(SELECT 1 FROM sys.objects WHERE type_desc = 'SQL_STORED_PROCEDURE' AND name = 'addExtraToEstada')
    DROP PROCEDURE dbo.addExtraToEstada;
GO
CREATE PROCEDURE dbo.addExtraToEstada @idExtra INT, @idEstada INT AS
    SET NOCOUNT ON
    BEGIN TRY
        BEGIN TRANSACTION SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
            DECLARE @associado VARCHAR(10)
            DECLARE @preçoDia INT

            SELECT @associado = associado, @preçoDia = preçoDia FROM dbo.Extra WHERE id = @idExtra

            IF(@associado <> 'pessoa')
                THROW 51000, 'Extra não é de pessoal', 5
            ELSE
                INSERT INTO dbo.EstadaExtra(estadaId, extraId, preçoDia)
                    VALUES(@idEstada, @idExtra, @preçoDia)
        COMMIT
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT !=0
            ROLLBACK;
        THROW
    END CATCH
```



```
            ELSE
                INSERT INTO dbo.EstadaExtra(estadaId, extraId, preçoDia)
                    VALUES(@idEstada, @idExtra, @preçoDia)
        COMMIT
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT !=0
            ROLLBACK;
        THROW
    END CATCH

/************************************* Adicionar extra pessoal a uma Estada ************************************************************/
GO
IF EXISTS(SELECT 1 FROM sys.objects WHERE type_desc = 'SQL_STORED_PROCEDURE' AND name = 'createEstadaInTime')
    DROP PROCEDURE dbo.createEstadaInTime;
GO
CREATE PROCEDURE dbo.createEstadaInTime @NIFResponsável INT, @NIFHóspede INT, @tempoEstada INT, @tipoAlojamento VARCHAR(8), @lotação TINYINT, @idExtraPessoal INT, @idExtraAlojamento INT AS
    SET NOCOUNT ON
    BEGIN TRY
        BEGIN TRANSACTION SET TRANSACTION ISOLATION LEVEL SERIALIZABLE  -- durante a criação de uma estada não pode haver nenhuma alteração ao estado da base de dados devido em especial ao
            DECLARE @id INT
            EXEC dbo.createEstada @NIFResponsável, @tempoEstada, @id OUTPUT

            EXEC dbo.addAlojamento @tipoAlojamento, @lotação, @id

            EXEC dbo.addHóspede @NIFHóspede, @id

            EXEC dbo.addExtraToAlojamento @idExtraAlojamento, @id

            EXEC dbo.addExtraToEstada @idExtraPessoal, @id
        COMMIT
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT !=0
            ROLLBACK;
        THROW
    END CATCH

/************************************* Teste ************************************************************************************/
```

```
mpleted with errors.
```

DESKTOP-61H5GUU (13.0 SP1) | DESKTOP-61H5GUU\rui_l ... | Glampin

Para responder aos requerimentos da alínea, e identificar cada sub-processamento individualmente foram criados seis procedimentos, cinco que correspondem a cada um dos sub-processamentos, e um que executa os outros e controla o fluxo da criação de uma estada. O procedimento *createEstada*, insere um tuplo na tabela Estada e outro na tabela HóspedeEstada, com os requisitos previamente enumerados, e uma vez que são realizados inserts na base de dados o nível de isolamento do procedimento é READ COMMITED, dado que os insert já têm um lock próprio e assim é reposto o nível de isolamento por omissão da base de dados. O procedimento *addAlojamento*, através de um select retira valores de atributos que serão usados no insert, de acordo com o tipo de alojamento, a lotação e a data e depois faz dois inserts, um na tabela AlojamentoEstada e outro na tabela EstadaExtra, e uma vez que nos inserts os dados a inserir dependem do resultado do select, é necessário garantir que as tabelas em uso não são alteradas, para não ocorrer lost updates, nem dirty reads, por isso foi definido que o nível de isolamento do procedimento seria REPEATABLE READ.

Para adicionar um hóspede a uma estada, foi criado o procedimento *addHóspede*, que executa apenas um insert na tabela HóspedeEstada, sendo por isso devido como nível de isolamento do procedimento READ COMMITED.

De modo a adicionar extras do tipo alojamento, ao alojamento associado à estada que está a ser criada foi criado o procedimento *addExtraToAlojamento,* que verifica se o extra a ser inserido é do tipo alojamento e em caso afirmativo é retirado os atributos em falta para o insert através de um select da tabela AlojamentoEstada, e no fim é feito o insert. Ora como o sucesso ou não do procedimento depende de as tabelas sobre as quais é feito os selects não alterarem até ao insert foi definido como nível de isolamento do procedimento REPEATABLE READ, para evitar lost updates e dirty reads. Para adicionar extras do tipo pessoa foi definido o procedimento *addExtraToEstada*, que tem um funcionamento semelhante ao procedimento anterior, alterando apenas a tabela na qual é feita o insert, assim este procedimento tem igualmente o nível de isolamento REPEATABLE READ, pelos mesmo motivos que o procedimento anterior.

Por fim, foi criado o procedimento *createEstadaInTime*, que executa os procedimentos anteriores, e uma vez que queremos que o estado da base de dados se mantenha o mesmo do início ao fim da execução do procedimento, o seu nível de isolamento foi definido como SERIALIZABLE.

i) Inscrever um hóspede na atividade

```
DESKT...61H5GUU\rui_l (61))
**  Objectivo           O objectivo deste script é o inscrever um hóspede numa actividade
**
**  Criado por          Grupo 4
**  Data de Criação     09/11/17
**
*********************************************************************************/
USE Glampinho

/************************************* Inscrever hóspede a actividade  ********************************************************************************/
GO
IF EXISTS(SELECT 1 FROM sys.objects WHERE type_desc = 'SQL_STORED_PROCEDURE' AND name = 'inscreverHóspede')
    DROP PROCEDURE dbo.inscreverHóspede;
GO
CREATE PROCEDURE dbo.inscreverHóspede @NIFHóspede INT, @númeroSequencial INT, @nomeParque NVARCHAR(30), @ano INT AS
    SET NOCOUNT ON
    BEGIN TRY
        BEGIN TRANSACTION SET TRANSACTION ISOLATION LEVEL REPEATABLE READ   -- para nao perdermos uma atualização
            DECLARE @dataRealização DATETIME2
            SELECT @dataRealização = dataRealização FROM dbo.Actividades WHERE nomeParque = @nomeParque AND númeroSequencial = @númeroSequencial AND ano = @ano

            IF NOT EXISTS (SELECT 1 FROM dbo.HóspedeEstada as hosEst INNER JOIN dbo.Estada as Est ON hosEst.id = Est.id JOIN dbo.AlojamentoEstada as AlojEst ON AlojEst.id = Est.id
                           WHERE hosEst.NIF = @NIFHóspede AND Est.dataFim > @dataRealização AND Est.dataInício < @dataRealização AND AlojEst.nomeParque = @nomeParque AND Est.dataFim > GETDATE()

                THROW 51000, 'Hóspede é inválido', 1

            INSERT INTO dbo.Paga(nomeParque, númeroSequencial, ano, NIF, preçoParticipante)
                SELECT @nomeParque, @númeroSequencial, @ano, @NIFHóspede, preçoParticipante FROM dbo.Actividades WHERE nomeParque = @nomeParque AND númeroSequencial = @númeroSequencial
        COMMIT
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT !=0
            ROLLBACK;
        THROW
    END CATCH

/************************************* Teste ********************************************************************************/
GO
INSERT INTO dbo.ParqueCampismo(nome, morada, estrelas, email)
    VALUES ('Glampinho', 'campo dos parques', 3, 'parque1@email.com')

INSERT INTO dbo.Hóspede(NIF, nome, morada, email, númeroIdentificação)
    VALUES (112233445, 'Teste', 'Rua teste', 'teste@teste.com', 11223344),
           (566778899, 'Maria', 'Rua 2', 'maria@gmail.com', 55667788)
```

É feita a verificação se o hóspede que deseja inscrever-se numa atividade tem as condições necessárias para esse efeito, nomeadamente se o hóspede está alojado no parque passado como parâmetro. Para isso é verificado em todas as estadas daquele parque se existe algum cliente hospedado num alojamento com o NIF passado como parâmetro e se sim, verificasse se a estada ainda está a decorrer, e termina depois da data de realização da actividade. Caso o resultado seja positivo, então basta inserir na tabela Paga a informação de que determinado hóspede realizou a atividade com númeroSequencial X, sendo que o preço da altura era Y. Para isso foi criado o procedimento *inscreverHóspede*, que depois de validar se o hóspede é válido para participar na actividade é inserido na tabela paga. Como não queremos que depois da validação do hóspede e antes deste ser inscrito na actividade, haja alterações à base de dados o procedimento tem nível de isolamento REPEATABLE READ

RESULTADO ESPERADO:

```
EXEC dbo.inscreverHóspede 566778899,2017,1, 'Glampinho'

SELECT * FROM dbo.Actividades
SELECT * FROM dbo.Paga
```

esults | Messages

nomeParq...	númeroSequen...	ano	nome	descrição	lotaçãoMáxi...	preçoParticipa...	dataRealização
Glampinho	1	2017	FUT7	Jogo de futebol 7vs7	14	3	2017-03-15 10:30:00.0000000

nomeParq...	númeroSequen...	ano	NIF	preçoParticipa...
Glampinho	1	2017	566778899	3

j) Proceder ao pagamento devido por uma estada, com emissão da respetiva fatura;

```
DESKT...61H5GUU\rui_l (59))*
**  Objectivo          O objectivo deste script é o de proceder ao pagamento
**                      devido por uma estada, emitindo a respectiva factura.
**                     Para isso, deve-se calcular o valor a pagar pelo
**                      alojamento, as actividades e os extras.
**
**  Criado por         Grupo 4
**  Data de Criação    14/11/17
**
*********************************************************************************/
USE Glampinho

/************************************* Retirar preço base  ***********************************************************************/
GO
IF EXISTS(SELECT 1 FROM sys.objects WHERE type_desc = 'SQL_STORED_PROCEDURE' AND name = 'getAlojamentoPreço')
    DROP PROCEDURE dbo.getAlojamentoPreço;
GO
CREATE PROCEDURE dbo.getAlojamentoPreço @idEstada INT, @idFactura INT, @ano INT, @linha INT, @novaLinha INT OUTPUT AS
    SET NOCOUNT ON
    BEGIN TRY
        BEGIN TRANSACTION SET TRANSACTION ISOLATION LEVEL READ COMMITTED
            INSERT INTO dbo.Item(idFactura, ano, linha, quantidade, preço, descrição, tipo)
                SELECT @idFactura, @ano, ROW_NUMBER() OVER (ORDER BY descrição) + @linha, 1, AlojEst.preçoBase, Aloj.descrição, 'alojamento'
                    FROM dbo.AlojamentoEstada AS AlojEst JOIN  dbo.Alojamento As Aloj ON AlojEst.localização = Aloj.localização AND AlojEst.nomeParque = Aloj.nomeParque WHERE id = @idEstad

            SELECT @novaLinha = @@ROWCOUNT + @linha
        COMMIT
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT !=0
            ROLLBACK;
        THROW
    END CATCH

/************************************* Retirar preço total dos extras de alojamento ***************************************************************************/
GO
IF EXISTS(SELECT 1 FROM sys.objects WHERE type_desc = 'SQL_STORED_PROCEDURE' AND name = 'getEstadaExtrasPreço')
    DROP PROCEDURE dbo.getEstadaExtrasPreço;
GO
CREATE PROCEDURE dbo.getEstadaExtrasPreço @idEstada INT, @idFactura INT, @ano INT, @linha INT, @novaLinha INT OUTPUT AS -- vai buscar o total a pagar de acordo com os extras para estada
    SET NOCOUNT ON
    BEGIN TRY
```

```
        COMMIT
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT !=0
            ROLLBACK;
        THROW
    END CATCH

/************************************* Retirar preço total dos extras de alojamento ********************************************************************************/
GO
IF EXISTS(SELECT 1 FROM sys.objects WHERE type_desc = 'SQL_STORED_PROCEDURE' AND name = 'getEstadaExtrasPreço')
    DROP PROCEDURE dbo.getEstadaExtrasPreço;
GO
CREATE PROCEDURE dbo.getEstadaExtrasPreço @idEstada INT, @idFactura INT, @ano INT, @linha INT, @novaLinha INT OUTPUT AS -- vai buscar o total a pagar de acordo
    SET NOCOUNT ON
    BEGIN TRY
        BEGIN TRANSACTION SET TRANSACTION ISOLATION LEVEL READ COMMITTED
            DECLARE @totalDias INT

            SELECT @totalDias = DATEDIFF(DAY, dataInício, dataFim) FROM dbo.Estada WHERE id = @idEstada

            INSERT INTO dbo.Item(idFactura, ano, linha, quantidade, preço, descrição, tipo)
                SELECT @idFactura, @ano, ROW_NUMBER() OVER (ORDER BY Ext.descrição) + @linha, @totalDias, EstExt.preçoDia * @totalDias, Ext.descrição, 'extra'
                    FROM dbo.EstadaExtra AS EstExt JOIN dbo.Extra AS Ext ON EstExt.extraId = Ext.id
                    WHERE EstExt.estadaId = @idEstada AND Ext.associado = 'alojamento'

            SELECT @novaLinha = @@ROWCOUNT + @linha
        COMMIT
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT !=0
            ROLLBACK;
        THROW
    END CATCH

/************************************* Retirar preço total dos extras de alojamento ********************************************************************************/
GO
IF EXISTS(SELECT 1 FROM sys.objects WHERE type_desc = 'SQL_STORED_PROCEDURE' AND name = 'getPessoalExtrasPreço')
```

```
on1
DESKT...61H5GUU\rui_l (59))*
                    FROM dbo.EstadaExtra AS EstExt JOIN dbo.Extra AS Ext ON EstExt.extraId = Ext.id
                    WHERE EstExt.estadaId = @idEstada AND Ext.associado = 'alojamento'

            SELECT @novaLinha = @@ROWCOUNT + @linha
        COMMIT
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT !=0
            ROLLBACK;
        THROW
    END CATCH

/************************************* Retirar preço total dos extras de alojamento ********************************************************************************/
GO
IF EXISTS(SELECT 1 FROM sys.objects WHERE type_desc = 'SQL_STORED_PROCEDURE' AND name = 'getPessoalExtrasPreço')
    DROP PROCEDURE dbo.getPessoalExtrasPreço;
GO
CREATE PROCEDURE dbo.getPessoalExtrasPreço @idEstada INT, @idFactura INT, @ano INT, @linha INT, @novaLinha INT OUTPUT AS  -- vai buscar o total a pagar de acordo com os extras para pessoal
    SET NOCOUNT ON
    BEGIN TRY
        BEGIN TRANSACTION SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
            DECLARE @totalDias INT
            DECLARE @totalHóspedes INT

            SELECT @totalDias = DATEDIFF(DAY, dataInício, dataFim) FROM dbo.Estada WHERE id = @idEstada

            SELECT @totalHóspedes = COUNT(NIF) FROM dbo.HóspedeEstada WHERE id = @idEstada

            INSERT INTO dbo.Item(idFactura, ano, linha, quantidade, preço, descrição, tipo)
                SELECT @idFactura, @ano, ROW_NUMBER() OVER (ORDER BY Ext.descrição) + @linha, @totalDias * @totalHóspedes, EstExt.preçoDia * @totalDias * @totalHóspedes, Ext.descrição, 'extra'
                    FROM dbo.EstadaExtra AS EstExt JOIN dbo.Extra Ext ON EstExt.extraId = Ext.id WHERE EstExt.estadaId = @idEstada AND Ext.associado = 'pessoa'

            SELECT @novaLinha = @@ROWCOUNT + @linha
        COMMIT
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT !=0
            ROLLBACK;
        THROW
    END CATCH

/************************************* Retirar preco total das actividade ********************************************************************************/
```

```
    BEGIN CATCH
        IF @@TRANCOUNT !=0
            ROLLBACK;
        THROW
    END CATCH

/************************************* Retirar preço total das actividade ********************************************************************************/
GO
IF EXISTS(SELECT 1 FROM sys.objects WHERE type_desc = 'SQL_STORED_PROCEDURE' AND name = 'getCustoTotalActividades')
    DROP PROCEDURE dbo.getCustoTotalActividades;
GO
CREATE PROCEDURE dbo.getCustoTotalActividades @idEstada INT, @idFactura INT, @ano INT, @linha INT AS    -- vai buscar o custo total das actividades
    SET NOCOUNT ON
    BEGIN TRY
        BEGIN TRANSACTION SET TRANSACTION ISOLATION LEVEL READ COMMITTED
            INSERT INTO dbo.Item(idFactura, ano, linha, quantidade, preço, descrição, tipo)
                SELECT @idFactura, @ano, ROW_NUMBER() OVER (ORDER BY Act.descrição) + @linha, COUNT(Paga.númeroSequencial), Paga.preçoParticipante * COUNT(Paga.númeroSequencial)
                    Act.descrição, 'actividade' FROM dbo.HóspedeEstada AS HosEst JOIN dbo.Paga ON HosEst.NIF = Paga.NIF JOIN dbo.Actividades as Act
                        ON Paga.nomeParque = Act.nomeParque AND Paga.númeroSequencial = Act.númeroSequencial WHERE HosEst.id = @idEstada
                        GROUP BY Act.descrição, Paga.preçoParticipante
        COMMIT
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT !=0
            ROLLBACK;
        THROW
    END CATCH

/************************************* Adicionar preço total à factura ********************************************************************************/
GO
IF EXISTS(SELECT 1 FROM sys.objects WHERE type_desc = 'SQL_STORED_PROCEDURE' AND name = 'addPreçoTotal')
    DROP PROCEDURE dbo.addPreçoTotal;
GO
CREATE PROCEDURE dbo.addPreçoTotal @idFactura INT, @ano INT AS
    SET NOCOUNT ON
    BEGIN TRY
        BEGIN TRANSACTION SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
            DECLARE @preçoTotal INT

            SELECT @preçoTotal = SUM(preço) FROM Item WHERE idFactura = @idFactura AND ano = @ano
```

```
    END CATCH

/************************************* Adicionar preço total à factura ********************************
GO
IF EXISTS(SELECT 1 FROM sys.objects WHERE type_desc = 'SQL_STORED_PROCEDURE' AND name = 'addPreçoTotal')
    DROP PROCEDURE dbo.addPreçoTotal;
GO
CREATE PROCEDURE dbo.addPreçoTotal @idFactura INT, @ano INT AS
    SET NOCOUNT ON
    BEGIN TRY
        BEGIN TRANSACTION SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
            DECLARE @preçoTotal INT

            SELECT @preçoTotal = SUM(preço) FROM Item WHERE idFactura = @idFactura AND ano = @ano

            UPDATE dbo.Factura SET preçoTotal = @preçoTotal WHERE id = @idFactura AND ano = @ano
        COMMIT
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT !=0
            ROLLBACK;
        THROW
    END CATCH

/************************************* Terminar Estada e apresentar factura ***********************
```

```
GO
IF EXISTS(SELECT 1 FROM sys.objects WHERE type_desc = 'SQL_STORED_PROCEDURE' AND name = 'finishEstadaWithFactura')
    DROP PROCEDURE dbo.finishEstadaWithFactura;
GO
CREATE PROCEDURE dbo.finishEstadaWithFactura @idEstada INT AS
    SET NOCOUNT ON
    BEGIN TRY
        BEGIN TRANSACTION SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
            DECLARE @NIFResponsável INT
            DECLARE @idFactura INT
            DECLARE @ano INT
            DECLARE @novaLinha INT
            DECLARE @data DATE
            DECLARE @nomeResponsável NVARCHAR(30)

            IF NOT EXISTS (SELECT 1 FROM dbo.Estada WHERE id = @idEstada)
                THROW 51000, 'A estada não existe', 1

            SELECT @ano = YEAR(GETDATE())

            SELECT @NIFResponsável = Hosp.NIF, @nomeResponsável = Hosp.nome FROM dbo.HóspedeEstada AS HospEst JOIN dbo.Hóspede AS Hosp
                ON HospEst.NIF = Hosp.NIF WHERE HospEst.id = @idEstada AND HospEst.hóspede = 'true'

            SELECT @idFactura = COUNT(id) + 1 FROM dbo.Factura WHERE ano = @ano

            INSERT INTO dbo.Factura(id, ano, NIFHóspede, nomeHóspede, preçoTotal)
                VALUES (@idFactura, @ano, @NIFResponsável, @nomeResponsável, 0)

            EXEC dbo.getAlojamentoPreço @idEstada, @idFactura, @ano, 0, @novaLinha OUTPUT

            EXEC dbo.getEstadaExtrasPreço @idEstada, @idFactura, @ano, @novaLinha, @novaLinha OUTPUT

            EXEC dbo.getPessoalExtrasPreço @idEstada, @idFactura, @ano, @novaLinha, @novaLinha OUTPUT

            EXEC dbo.getCustoTotalActividades @idEstada, @idFactura, @ano, @novaLinha

            EXEC dbo.addPreçoTotal @idFactura, @ano
        COMMIT
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT !=0
            ROLLBACK;
        THROW
    END CATCH
```

Para a criação da factura, é necessário, inserir na tabela Item correspondente à factura a ser criada todos os itens que tenham um custo da estada, para isso foram criados seis procedimentos armazenados, cinco que calculam o preço a pagar por cada item, tendo em consideração a quantidade e um que cria a factura e executa os outros. Assim, começou-se por criar o procedimento *getAlojamentoPreço*, que insere em item o preço a pagar pelos alojamentos usados pela estada que vai ser paga, uma vez que neste procedimento apenas se realiza um insert e como estes têm o seu próprio lock, a transação tem o nível de isolamento READ COMMITTED. Para saber quanto é necessário pagar referente aos extra do tipo alojamento usufruídos pela estada, foi criada o procedimento *getEstadaExtrasPreço*, que insere no item todos os extras do tipo alojamento usados e indica o preço a pagar, tendo em conta o preço por dia do extra e o número de dias que durou a estada. Uma vez que neste procedimento a instrução a executar é um insert a transação tem o nível de isolamento READ COMMITTED.

Para calcular o custo total dos extras do tipo pessoal usufruídos no decorrer da estada, foi criado o procedimento *getPessoalExtrasPreço*, que após calcular quantos hóspedes estavam na estada, e quando dias durou a estada, insere em item os extras de pessoal usado pelos hóspedes da estada, sendo que o preço a pagar por cada extra corresponde ao produto do preço por dia do extra ao número de dias da estada e ao número de hóspedes da estada. A transação deste procedimento tem o nível de isolamento REPEATABLE READ, uma vez que não queremos que seja removido ou adicionado nenhum hóspede à estada enquanto calculamos quanto o hóspede responsável deve pagar pelos extras

Para saber quanto se devia pagar referente às actividades, foi criado o procedimento *getCustoTotalActividades*, que ao inserir todas as actividades que hóspedes da estada participaram indica o preço a pagar correspondente a cada actividade que corresponde ao produto do preço por participante da actividade pelo número de participantes que eram hóspedes na estada. Uma vez que na transação deste procedimento apenas é realizado um insert, o seu nível de isolamento é READ COMMITTED.

Após o calculo de todos os items da estada, é necessário calcular o preço total a pagar pela estada e inseri-lo na factura, para isso foi criado o procedimento *addPreçoTotal*, que calcula o preço total, somando os preços individuais dos items da factura previamente calculados e actualiza a factura colocado no tuplo correspondente o total a pagar. Uma vez que o valor do atributo a actualizar na factura depende do calculo feito através do select e como não queremos permitir que após o calculo do preço a pagar haja alteração das tabelas envolvidas a transação deste procedimento tem o nível de isolamento REPEATABLE READ.

Por fim, foi criado o procedimento *finishEstadaWithFactura*, que após validar que a estada para a qual irá ser criada a factura existe, cria a factura e executa os procedimentos anteriormente criados. Uma vez que queremos que não haja alteração ao estado da base de dados durante a execução da transação do procedimento o seu nível de isolamento foi definido para SERIALIZABLE.

RESULTADO ESPERADO: (nota: pagamento da fatura da estada 1)

```
EXEC dbo.finishEstadaWithFactura 1

SELECT * FROM Factura WHERE NIFHóspede=112233445
```

100 %

Results | Messages

	id	ano	nomeHóspe...	NIFHóspe...	preçoTo...
1	1	2017	José	112233445	1545
2	2	2017	Maria	2	7353
3	3	2017	Maria	123456789	50
4	4	2017	Maria	3	84
5	5	2017	Maria	4	1112
6	6	2017	NULL	NULL	NULL
7	7	2017	NULL	NULL	NULL

```
SELECT * FROM Item WHERE idFactura=1
```

100 %

Results | Messages

	idFactu...	ano	lin...	quantida...	preço	descrição
1	1	2017	1	1	15	vazia
2	1	2017	2	38	380	descricao
3	1	2017	3	76	1140	teste
4	1	2017	4	2	4	Jogo de futebol 5vs5
5	1	2017	5	2	6	Jogo de futebol 7vs7

k) Enviar emails a todos os hóspedes responsáveis por estadas que se irão iniciar dentro de um dado período temporal. Os emails dever ser enviados usando o procedimento armazenado SendMail que recebe o NIF do cliente e o texto da mensagem a enviar.

```
GO
CREATE PROCEDURE dbo.sendEmail @NIF INT, @email NVARCHAR(30), @text NVARCHAR(255) AS
        PRINT 'De: Gerência Glampinho'
        PRINT 'Para: ' + @email
        PRINT 'Cliente com o NIF: ' + CAST(@NIF AS VARCHAR)
        PRINT 'Mensagem: ' + @text
        PRINT ''

/************************************ Envia email a todos os hóspedes responsáveis por estadas a começar dentro de x temp **************
GO
CREATE PROCEDURE dbo.SendEmails @periodoTemporal INT AS
        DECLARE @NIF INT
        DECLARE @email NVARCHAR(30)

        DECLARE iterate_NIFs CURSOR LOCAL FORWARD_ONLY FOR
                SELECT Hosp.NIF, Hosp.email FROM dbo.Hóspede AS Hosp JOIN dbo.HóspedeEstada AS HospEst ON Hosp.NIF = HospEst.NIF
                        JOIN dbo.Estada AS Est ON HospEst.id = Est.id WHERE HospEst.hóspede = 'true' AND Est.dataInício <= DATEADD(DAY, @per
        OPEN iterate_NIFs

        FETCH NEXT FROM iterate_NIFs INTO @NIF, @email
        WHILE @@FETCH_STATUS = 0
                BEGIN
                        EXEC SendEmail @NIF, @email, 'A sua estada no Parque Glampinho está à sua espera! Para mais informações contacte-nos
                        FETCH NEXT FROM iterate_NIFs INTO @NIF, @email
                END
        CLOSE iterate_NIFs
        DEALLOCATE iterate_NIFs
```

Foi criado o procedimento armazenado SendEmails que recebe um período temporal. Este parâmetro referido indica os dias que faltam para se iniciar as estadas dos hóspedes em causa. Para essa verificação é feito um SELECT à base de dados na Estada de forma a ir buscar os clientes que se enquadrem entro de esse período temporal(***Est.dataInício <= DATEADD(DAY, @periodoTemporal, GETDATE())***).
Posteriormente é aberto um cursor para percorrer a tabela daí resultante de forma a enviar para cada um desses clientes o email desejado, através do chamamento do procedimento SendEmail.
O nível de isolamento é definido como REPEATABLE READ de forma a prevenir que entretanto ocorra alguma alteração na tabela fatura que modifique o valor da média.

RESULTADO ESPERADO:

```
INSERT INTO dbo.Estada(id, dataInício, dataFim)
    VALUES  (2, '2017-11-20', '2017-11-27'),
            (3, '2017-11-20', '2017-11-27'),
            (4, '2017-11-26', '2017-11-30')


INSERT INTO dbo.HóspedeEstada(NIF, id, hóspede)
    VALUES  (112233445, 2, 'true'),
            (566778899, 2, 'false')

EXEC dbo.sendEmails 7
```

```
00 %
Messages
De: Gerência Glampinho
Para: teste@teste.com
Cliente com o NIF: 112233445
Mensagem: A sua estada no Parque Glampinho está à sua espera! Para mais informações contacte-nos para glampinho@email.com.
```

l) Listar todas as atividades com lugares disponíveis para um intervalo de datas especificado;

Foi criada a função listAtividades que é responsável por retornar uma tabela que contem apenas as atividades que se encontram disponíveis e dentro do intervalo passado como parâmetro.

Para isso é feito um SELECT à base de dados na tabela Atividades de forma a ir buscar as atividades que tem uma data de realização dentro desse intervalo. De seguida é preciso contar o número de participantes dessas atividades de forma a garantir que têm lugares disponíveis para a eventual inserção de um cliente nessa mesma atividade.

```
drop function dbo.listAtividades
drop proc listarAtividades
GO
CREATE FUNCTION dbo.listAtividades(@dataInicio date,@dataFim date)
RETURNS  @rtnTable TABLE
(
    -- columns returned by the function
    nome nvarchar(30) NOT NULL,
    descrição nvarchar(255) NOT NULL
)
AS
BEGIN
DECLARE @partipantes int
DECLARE @TempTable table (nome nvarchar(30),descrição nvarchar(255))

insert into @TempTable
SELECT nome,descrição FROM Actividades INNER JOIN (
        SELECT ano,númeroSequencial,count(númeroSequencial) as participantes FROM Paga
        GROUP BY númeroSequencial,ano ) as A ON A.númeroSequencial=Actividades.númeroSequencial and A.ano=Actividades.ano and lotaçãoMáxima

--This select returns data
insert into @rtnTable
SELECT nome,descrição FROM @TempTable
return
END
GO
```

Por fim foi criado um procedimento armazenado que chama essa função.

O nível de isolamento deste procedimento armazenado é REPEATABLE READ para se ter a garantia que durante o intervalo entre primeiro select e o segundo não vá haver alterações naquilo que é apresentado, visto que entretanto podeira ocorrer uma inserção de um hóspede na atividade selecionada e perante essa situação estaríamos a perder um eventual update.

```
/************************************ Apresenta todas as actividades com lugares disponiveis para um in
GO
IF EXISTS(SELECT 1 FROM sys.objects WHERE TYPE_DESC = 'SQL_STORED_PROCEDURE' AND NAME = 'listarAtividades'
    DROP PROCEDURE dbo.listarAtividades;
GO
CREATE PROCEDURE dbo.listarAtividades @dataInicio DATETIME2, @dataFim DATETIME2 AS
    SET NOCOUNT ON
    BEGIN TRY
        BEGIN TRANSACTION SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

            SELECT * FROM dbo.listAtividades(@dataInicio, @dataFim)
            IF @@ROWCOUNT = 0
                THROW 51000, 'As actividades encotram-se lotadas ou fora do intervalo especificado', 1
        COMMIT

    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT !=0
            ROLLBACK;
        THROW
    END CATCH

/************************************ Teste ***************************************************************
GO
```

RESULTADO ESPERADO:

```
exec listarAtividades '2016-03-12' ,'2017-03-16'

INSERT INTO Paga(ano,NIF,nomeParque,númeroSequencial,preçoParticipante)
    VALUES (2017,2,'Glampinho',2,2)

INSERT INTO dbo.Hóspede(NIF, nome, morada, email, númeroIdentificação)
    VALUES (2, 'José', 'Rua 1', 'OI@gmail.com', 11223344)

INSERT INTO Actividades (nomeParque,númeroSequencial,ano, nome, descrição, lotaçãoMáxima, preçoParticipante, dataRealização)
    VALUES('Glampinho',2,2017,'Yoga','Relaxing',2,3,'04-15-16 10:30')
```

sults | Messages

nome	descriç...
Yoga	Relaxing

m) Obter a média dos pagamentos realizados num dado ano, calculada com um intervalo de amostragem especificado;

Foi criado um procedimento onde é aberto um cursor que tem como objetivo iterar de N em N sobre a tabela Fatura, sendo esse N recebido como parâmetro. Para isso foi utilizado uma propriedade do cursor denominada RELATIVE que permite avançar para a "N rows" relativamente à coluna selecionada. É assumido que nunca é passado como parâmetro um N igual ou superior ao numero de amostras total na tabela Fatura. Foram também declaradas 3 variáveis:

- Uma para ir guardando o valor total dos pagamentos selecionados;
- Outra para guardar o valor atual do pagamento da Fatura selecionada.
- Uma variável para ir guardando o número de amostras selecionadas.

```
/************************************* CALCULA A MÉDIA DOS PAGAMENTOS REALIZADOS NUM DADO ANO ************************************************************/
GO
IF EXISTS(SELECT 1 FROM sys.objects WHERE TYPE_DESC = 'SQL_STORED_PROCEDURE' AND NAME = 'mediaPagamentos')
    DROP PROCEDURE dbo.mediaPagamentos;
GO
CREATE PROCEDURE dbo.mediaPagamentos @n INT AS
    SET NOCOUNT ON
    BEGIN TRY
        BEGIN TRANSACTION SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

            DECLARE @valorTotal INT = 0
            DECLARE @num INT = 0
            DECLARE @valorPagamento INT

            DECLARE iterate_estada CURSOR LOCAL SCROLL FOR SELECT preçoTotal FROM Factura

            OPEN iterate_estada
            FETCH NEXT FROM iterate_estada INTO @valorPagamento

            WHILE @@FETCH_STATUS = 0
            BEGIN
                SET @valorTotal = @valorPagamento + @valorTotal
                SET @num = @num + 1

                FETCH RELATIVE @n FROM iterate_estada INTO @valorPagamento
            END

            DECLARE @media INT = dbo.media (@valorTotal, @num)

            PRINT 'Média de pagamentos: ' + CAST(@media as VARCHAR(30))

            CLOSE iterate_estada
            DEALLOCATE iterate_estada
        COMMIT
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT !=0
            ROLLBACK;
        THROW
    END CATCH

/************************************* Teste ************************************************************/
```

Uma vez que durantea execução deste procedimento não queremos que haja alterações à base de dados, a transacção tem o nível de isolamento SERIALIZABLE.

Depois de calculado o valor total de pagamentos é chamada a função media que tem como objetivo calcular a média de pagamentos.

```
CREATE FUNCTION dbo.media (@preço INT,@amostras INT)
RETURNS INT
AS
BEGIN
DECLARE @MEDIA INT

SET @MEDIA = @preço/@amostras

RETURN @MEDIA
```

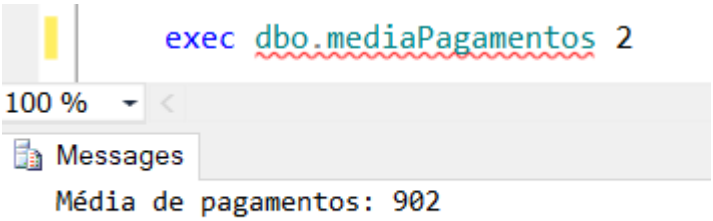
RESULTADO ESPERADO:

Faturas:

Results | Messages

	id	ano	nomeHóspe...	NIFHóspe...	preçoTo...
1	1	2017	José	112233445	1545
2	2	2017	Maria	2	7353
3	3	2017	Maria	123456789	50
4	4	2017	Maria	3	84
5	5	2017	Maria	4	1112

Média:



(1545+50+1112)/3=902

n) ***Criar a vista bungalows que permita executar as instruções SQL SELECT, INSERT, DELETE e UPDATE apenas sobre a parte dos alojamentos correspondente a bungalows. A vista deve produzir todas as colunas de um alojamento bungalow e todas as colunas de um parque. Os comandos INSERT, DELETE e UPDATE apenas alteram os dados relativos ao alojamento e não os relativos aos parques.***

Foram criados triggers para os INSERTS, DELETES e UPDATES visto que como a vista Bungalows é resultante da junção de várias tabelas, quando se insere/apaga ou actualiza algo da tabela é preciso inserir/eliminar ou atualizar também das tabelas de que a vista depende. Por essa razão um trigger INSTEAD OF é útil na medida quem que quando alguém tentar alterar diretamente através da vista, é realizada a operação que queremos especificada no trigger.

Ambos os triggers seguem a mesma lógica:

- Se não foi feito nenhum insert/delete, então o trigger não realiza nenhuma operaçãp;
- Caso tenha sido feito apenas um insert/delete é chamado o procedimento armazenado correspondente para inserir/apagar da tabela Alojamento e Bungalow.
- Se forem feitos mais do que um insert ao mesmo tempo, é preciso abrir um cursor para percorrer a tabela inserted. A cada iteração será inserido nas tabelas Alojamento e Bungalow os tuplos que o cursor iterou da tabela inserted, e assim sucessivamente até não existirem mais tuplos a inserir

TRIGGER DE INSERT

```
/*************** TRIGGER DE INSERT ****************************************************************/
GO
CREATE TRIGGER trgBungalowInsert
ON Bungalows
INSTEAD OF INSERT
AS
BEGIN TRANSACTION
BEGIN TRY
        SET nocount ON
                DECLARE @num int
                DECLARE @nomeParque NVARCHAR(30),@nome NVARCHAR(30),@localização NVARCHAR(30),@descrição NVARCHAR(30),@preçoBase INT,
            @númeroMáximoPessoas INT,@tipologia CHAR(2)
    SELECT @num=count(*) FROM inserted
    --Diferentes formas de inserir, dependendo do nº de inserções
    IF (@num = 0)
        RETURN
    IF (@num = 1)
    BEGIN
                select @nomeParque=nome,@nome=nomeAlojamento,@localização=localização,@descrição=descrição,@preçoBase=preçoBase,@númeroMáxi
                exec dbo.InsertAlojamentoBungalow @nomeParque,@nome,@localização,@descrição,@preçoBase,@númeroMáximoPessoas,@tipologia
        END

    ELSE
    BEGIN
        declare acursor cursor local
        for select nome,nomeAlojamento,localização,descrição,preçoBase,númeroMáximoPessoas,tipologia  from inserted
        open acursor
        fetch next from acursor into @nomeParque,@nome,@localização,@descrição,@preçoBase,@númeroMáximoPessoas,@tipologia
        while (@@FETCH_STATUS = 0 )
        BEGIN
        exec dbo.InsertAlojamentoBungalow @nomeParque,@nome,@localização,@descrição,@preçoBase,@númeroMáximoPessoas,@tipologia
        fetch next from acursor into @nomeParque,@nome,@localização,@descrição,@preçoBase,@númeroMáximoPessoas,@tipologia
                END
        END
        COMMIT
```

TRIGGER DE DELETE

```
/*************** TRIGGER DE DELETE ****************************************************************/

DROP TRIGGER trgBungalowDelete
GO
CREATE TRIGGER trgBungalowDelete
ON Bungalows
INSTEAD OF DELETE
AS
BEGIN TRANSACTION
BEGIN TRY
        SET nocount ON
                DECLARE @num int
                DECLARE @nomeParque NVARCHAR(30),@localização NVARCHAR(30)
    SELECT @num=count(*) FROM deleted
    --Diferentes formas de inserir, dependendo do nº de inserções
    IF (@num = 0)
        RETURN
    IF (@num = 1)
    BEGIN
                select @nomeParque=nome,@localização=localização from deleted
                exec dbo.deleteAlojamento @localização,@nomeParque
        END
        COMMIT
END TRY
BEGIN CATCH
        IF @@TRANCOUNT > 0 ROLLBACK;
END CATCH
```

TRIGGER DE UPDATE

```
/*------------ TRIGGER DE UPDATE ----------------------------------------------------------------*/
GO
IF EXISTS (SELECT 1 FROM sys.triggers WHERE NAME = 'trgBungalowUpdate')
   DROP TRIGGER dbo.trgBungalowUpdate
GO
CREATE TRIGGER dbo.trgBungalowUpdate ON dbo.Bungalows INSTEAD OF UPDATE AS
SET NOCOUNT ON
BEGIN TRY
    BEGIN TRANSACTION
        DECLARE @num INT
        DECLARE @nomeParqueDeleted NVARCHAR(30), @localizaçãoDeleted NVARCHAR(30)
        DECLARE @nomeParque NVARCHAR(30), @localização NVARCHAR(30), @tipologia CHAR(2)
        DECLARE @nome NVARCHAR(30), @descrição NVARCHAR(30), @preco INT, @númeroMáximoPessoas TINYINT

        SELECT @num = count(*) FROM deleted

        IF (@num = 0)
            RETURN

        IF (@num = 1)
        BEGIN
            SELECT @nomeParqueDeleted = nome, @localizaçãoDeleted = localização FROM deleted
            SELECT @nomeParque = nome, @localização = localização, @tipologia = tipologia,
               @nome = nome, @descrição = descrição, @preco = precoBase, @númeroMáximoPessoas = númeroMáximoPessoas FROM inserted

            UPDATE dbo.Bungalow SET localização = @localização, tipologia = @tipologia WHERE nomeParque = @nomeParqueDeleted AND localização = @localizaçãoDeleted
            UPDATE dbo.Alojamento SET localização = @localização, descrição = @descrição, precoBase = @preco, númeroMáximoPessoas = @númeroMáximoPessoas
                WHERE nomeParque = @nomeParqueDeleted AND localização = @localizaçãoDeleted
        END

        ELSE
            BEGIN
                DECLARE actualizaBungalowsDelete CURSOR LOCAL
                    FOR SELECT nome, localização FROM deleted

                DECLARE actualizaBungalowsInsert CURSOR LOCAL
                    FOR SELECT nome, localização, tipologia, nomeAlojamento, descrição, precoBase, númeroMáximoPessoas FROM inserted

                OPEN actualizaBungalowsDelete
                OPEN actualizaBungalowsInsert
                FETCH NEXT FROM actualizaBungalowsDelete INTO @nomeParqueDeleted, @localizaçãoDeleted

                WHILE (@@FETCH_STATUS = 0 )
                BEGIN
                    FETCH NEXT FROM actualizaBungalowsInsert INTO @nomeParque, @localização, @tipologia,
                        @nome, @descrição, @preco, @númeroMáximoPessoas

                    UPDATE dbo.Bungalow SET localização = @localização, tipologia = @tipologia WHERE nomeParque = @nomeParqueDeleted AND localização = @localizaçãoDeleted
                    UPDATE dbo.Alojamento SET localização = @localização, descrição = @descrição, precoBase = @preco, númeroMáximoPessoas = @númeroMáximoPessoas
                        WHERE nomeParque = @nomeParqueDeleted AND localização = @localizaçãoDeleted

                    FETCH NEXT FROM actualizaBungalowsDelete INTO @nomeParqueDeleted, @localizaçãoDeleted
                END

                CLOSE actualizaBungalowsDelete
                DEALLOCATE actualizaBungalowsDelete
                CLOSE actualizaBungalowsInsert
                DEALLOCATE actualizaBungalowsInsert
            END
    COMMIT
END TRY
BEGIN CATCH
    IF @@TRANCOUNT !=0
        ROLLBACK;
    THROW
END CATCH
```

EXEMPLO:

```
SELECT * FROM Bungalows
UPDATE Bungalows SET tipologia='T2' WHERE nome='Glampinho' and localização='Rua 6'
SELECT * FROM Bungalows
```

100 %

Results | Messages

	nome	email	estrel...	morada	descrição	localizaç...	nomeAlojamento	númeroMáximoPess...	preçoBa...	tipoAlojame...	tipolo...
1	Glampinho	parque1@email.com	3	campo dos parques	primeiro bungalow	Rua 6	Bungalow hoje	10	15	bungalow	T1
2	Glampinho	parque1@email.com	3	campo dos parques	segundo bungalow	Rua 7	Bungalow ontem	9	15	bungalow	T3

	nome	email	estrel...	morada	descrição	localizaç...	nomeAlojamento	númeroMáximoPess...	preçoBa...	tipoAlojame...	tipolo...
1	Glampinho	parque1@email.com	3	campo dos parques	primeiro bungalow	Rua 6	Bungalow hoje	10	15	bungalow	T2
2	Glampinho	parque1@email.com	3	campo dos parques	segundo bungalow	Rua 7	Bungalow ontem	9	15	bungalow	T3