***Sistemas de Informação II***

Turma LI52D | Inverno 2017/2018



**INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA**

***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)])
  + 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.

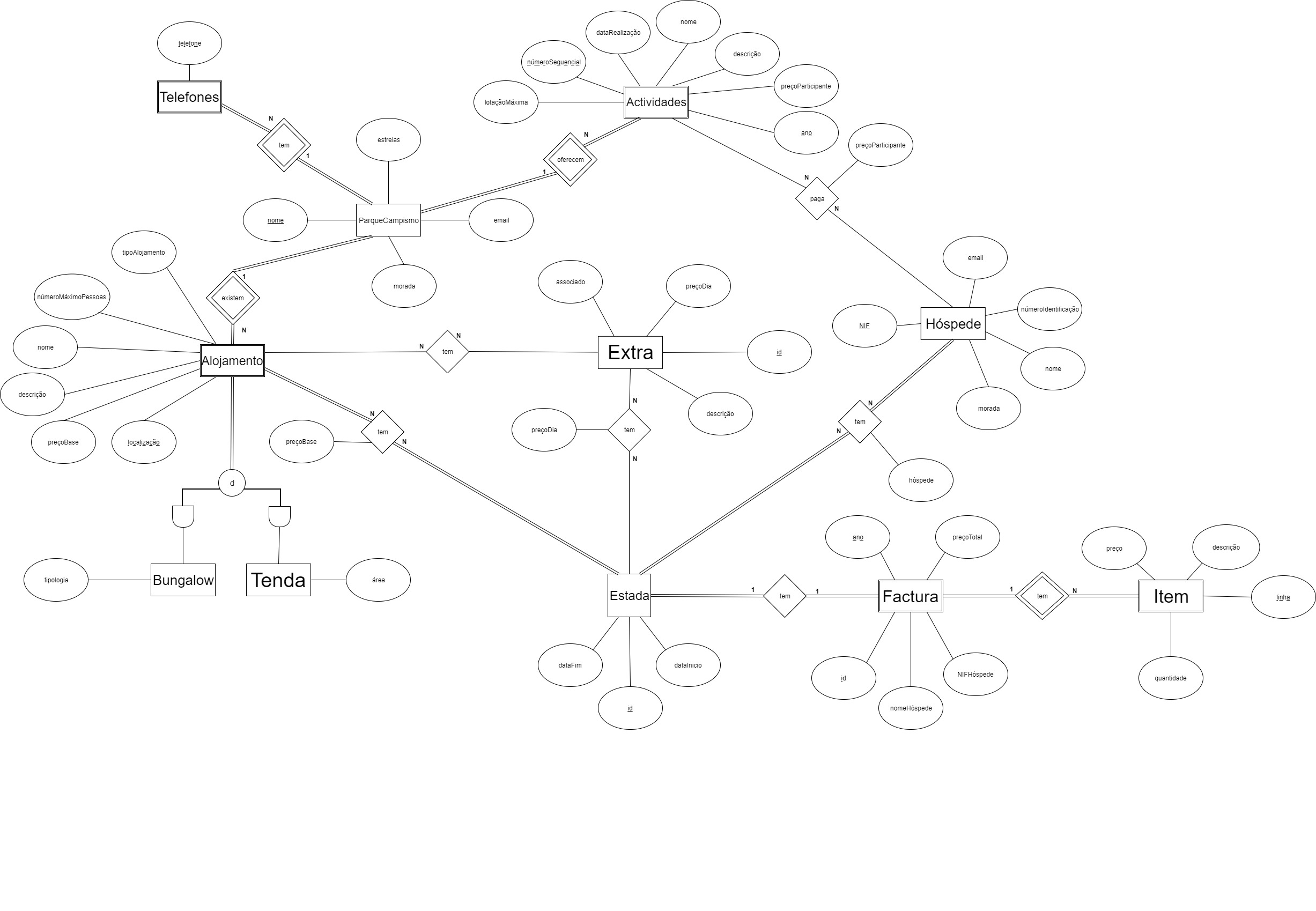
* ***Os mesmos hóspedes podem ficar alojados diversas vezes no mesmo alojamento, desde que em períodos diferentes.***
* ***Os extras para cada estada são definidos no momento do registo no sistema da estada.***

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 é 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

1. ***Criação do modelo físico.***

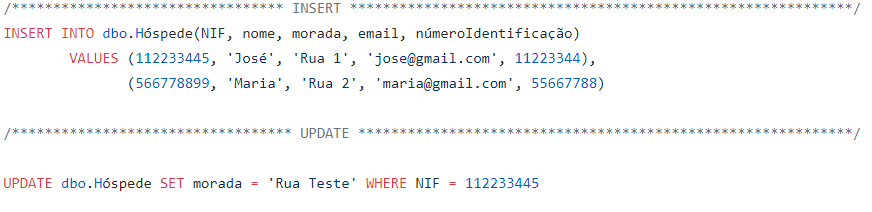
Foi realizado um script com a criação de todas as entidades necessárias, inclusive as associações entre elas, respeitando as restrições de integridade que são possíveis implementar em código SQL.

1. ***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

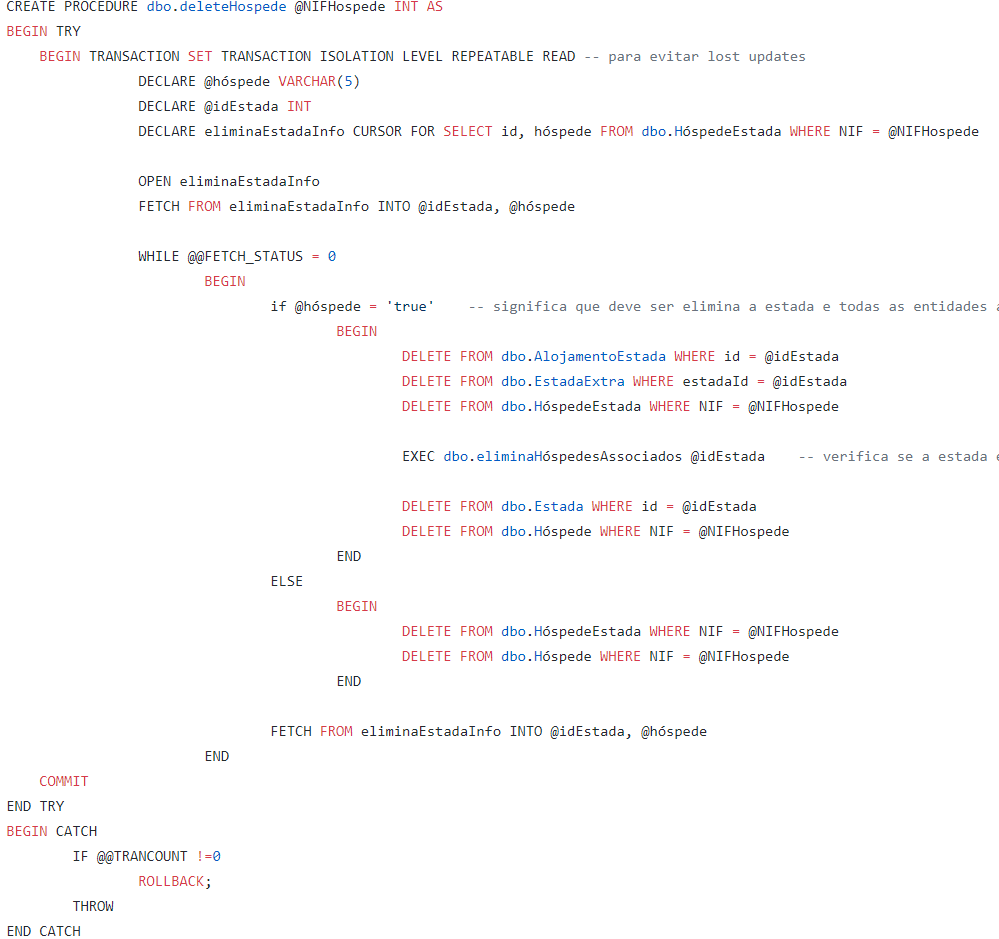
1. ***Inserir, remover e atualizar informação de um hóspede.***

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.



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.

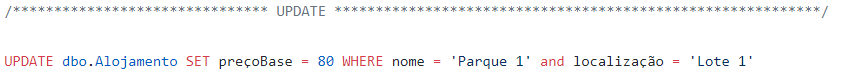


Uma imagem com captura de ecrã

Descrição gerada com confiança muito alta

1. ***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.

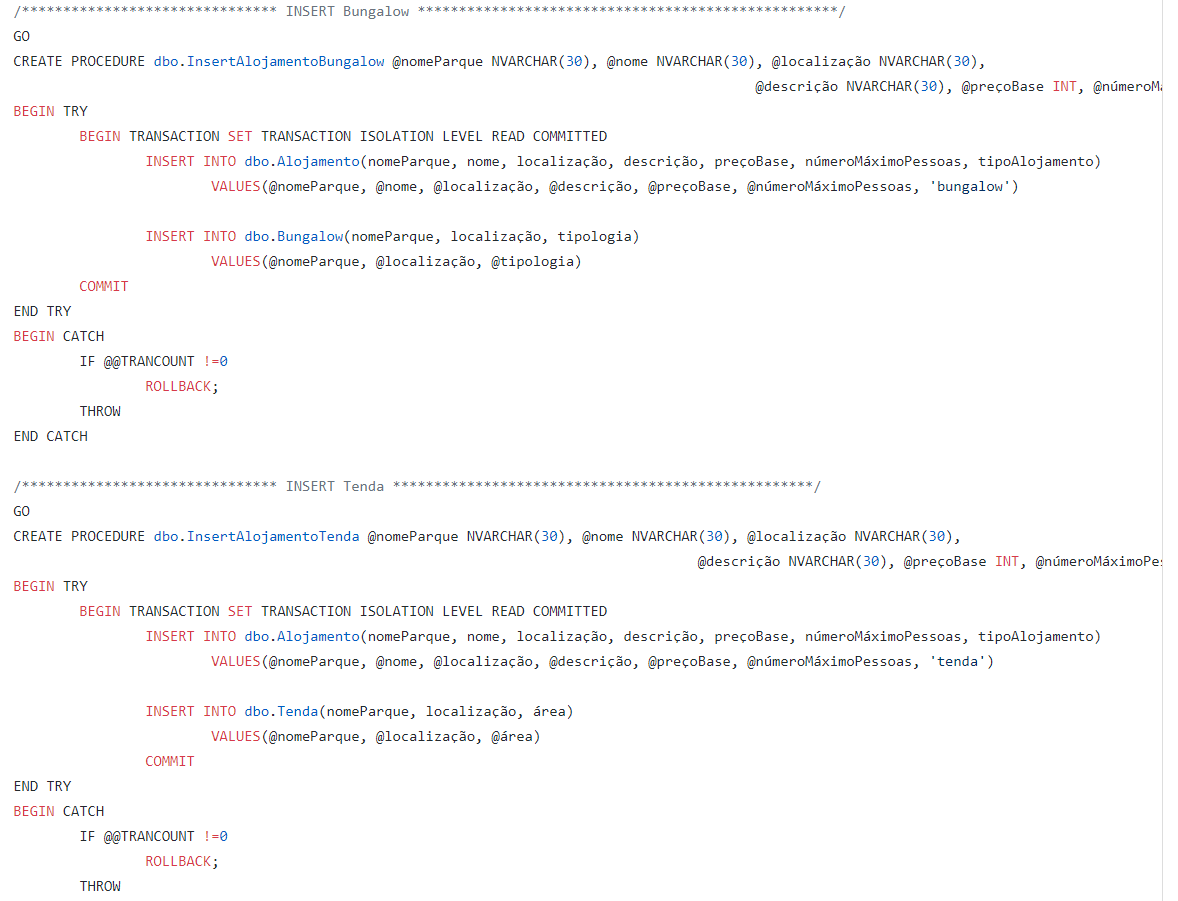


Para a inserção de um alojamento há que ter atenção os seguintes aspetos:

* 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



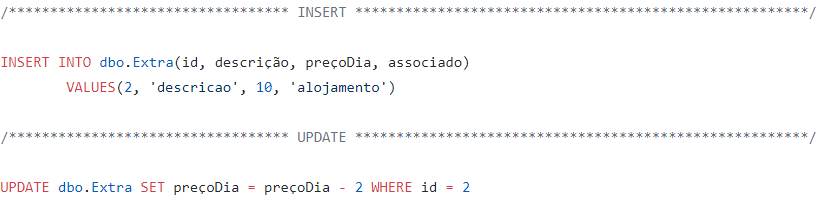
Para apagar um Alojamento é necessário apagá-lo primeiro das tabelas que representam as associações entre outras entidades e o alojamento com o nome e nome do Parque passados como parâmetro, e caso o alojamento seja o único de uma dada estada, também esta, bem como, os elementos da associação HóspedeEstada, 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.

Uma imagem com captura de ecrã

Descrição gerada com confiança muito alta

1. ***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.



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.

Uma imagem com captura de ecrã

Descrição gerada com confiança muito alta

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

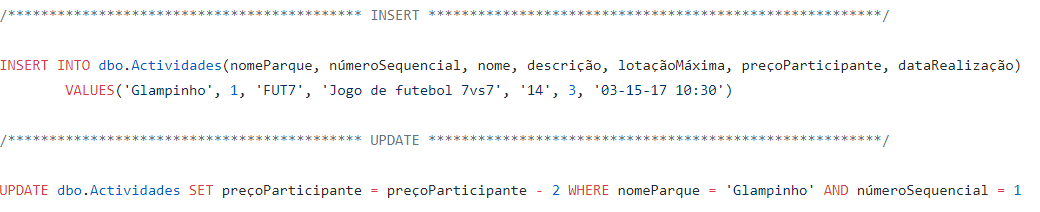
Esta alínea é muito semelhante à alínea anterior, sendo a única diferença, referente ao tipo de extra a que se refere, sendo neste caso extras do tipo pessoal

Uma imagem com captura de ecrã

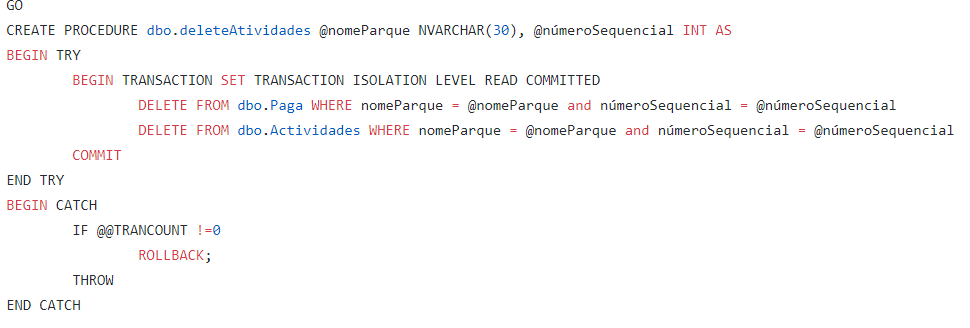
Descrição gerada com confiança muito alta

1. ***Inserir, remover e atualizar informação de uma atividade.***

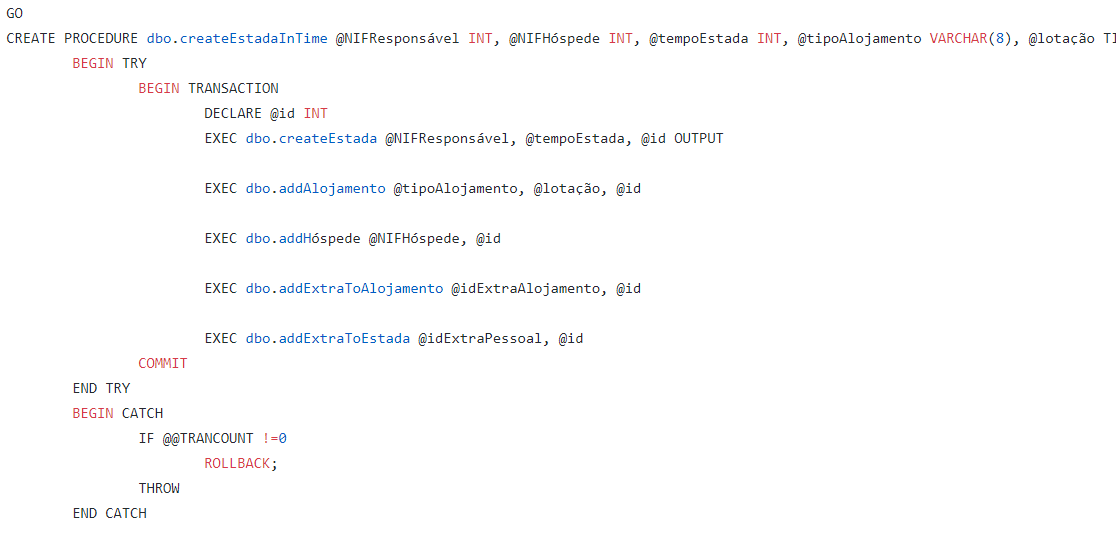
Não foi necessário a criação de um procedimento armazenado para inserir e atualizar a informação de um hóspede, visto que trata apenas de uma instrução de INSERT/UPDATE na respetiva entidade.



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.

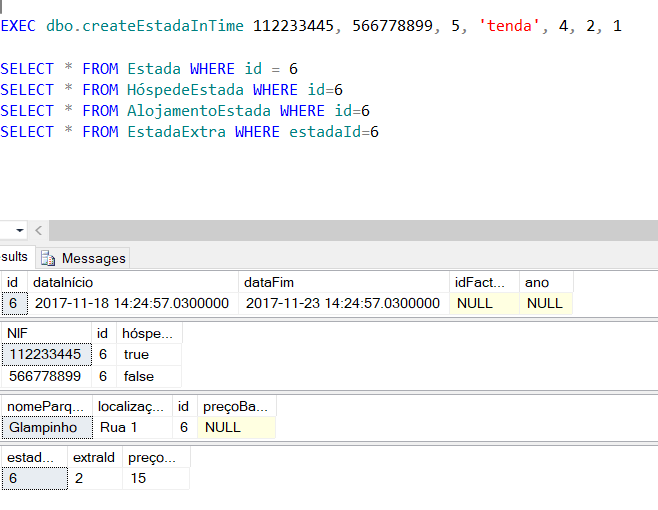


1. ***Criar uma estada para um dado período de tempo. Este processamento deve ser dividido nos seguintes sub-processamentos.***



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 )***



Uma imagem com captura de ecrã

Descrição gerada com confiança muito alta

Uma imagem com captura de ecrã

Descrição gerada com confiança muito alta

Uma imagem com captura de ecrã

Descrição gerada com confiança muito alta

Uma imagem com captura de ecrã

Descrição gerada com confiança muito alta

Uma imagem com captura de ecrã

Descrição gerada com confiança muito alta

Uma imagem com captura de ecrã

Descrição gerada com confiança muito alta

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.

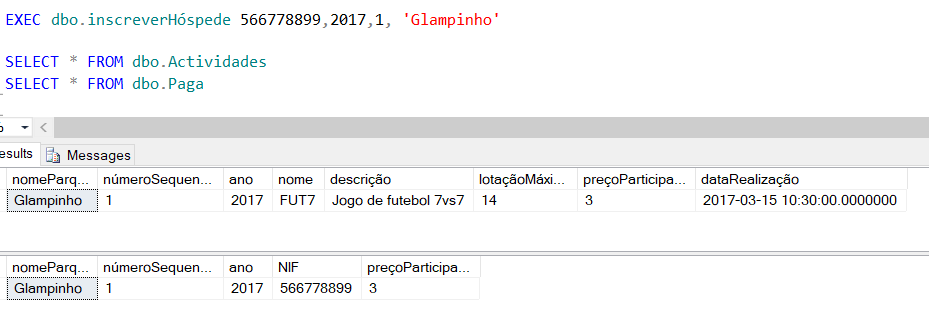
1. ***Inscrever um hóspede na atividade***

Uma imagem com captura de ecrã

Descrição gerada com confiança muito alta

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:***



1. ***Proceder ao pagamento devido por uma estada, com emissão da respetiva fatura;***

Uma imagem com captura de ecrã

Descrição gerada com confiança muito alta

Uma imagem com captura de ecrã

Descrição gerada com confiança muito alta

Uma imagem com captura de ecrã

Descrição gerada com confiança muito alta

Uma imagem com captura de ecrã

Descrição gerada com confiança muito alta

Uma imagem com captura de ecrã

Descrição gerada com confiança muito alta

Uma imagem com captura de ecrã

Descrição gerada com confiança muito alta

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.

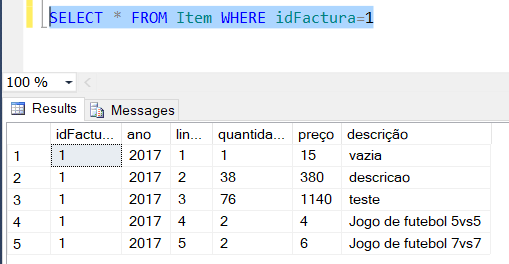
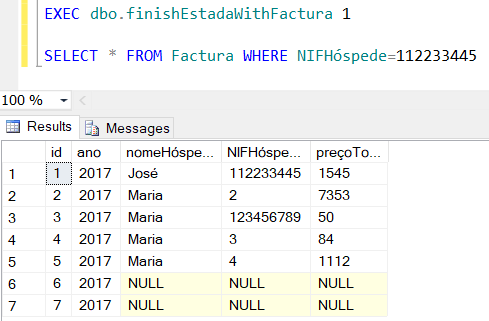
Para calcular o custo total dos extras do tipo pessoal usufruídos no decorrer da estada, foi criado o procedimento *getPessoalExtrasPreço*, que após calcular quantos hóspedes estavam na estada, e quando dias durou a estada, insere em item os extras de pessoal usado pelos hóspedes da estada, sendo que 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)***



1. ***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.***



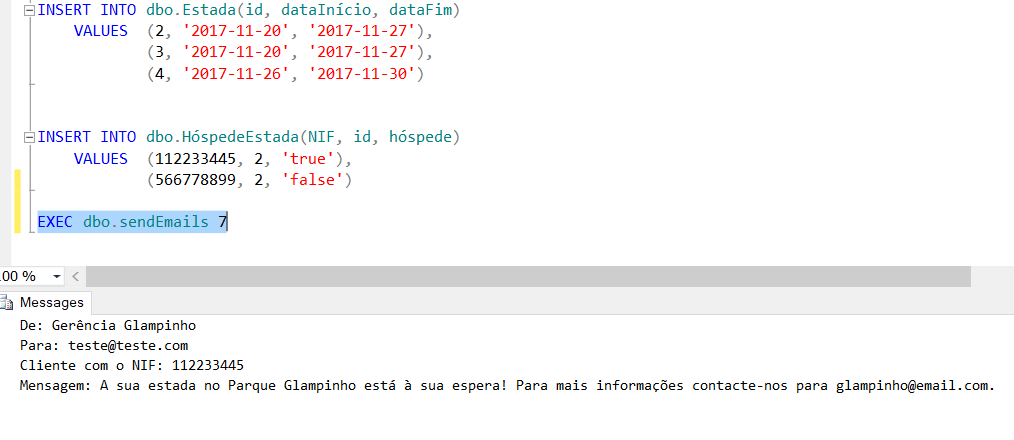
Foi criado o procedimento armazenado SendEmails que recebe um período temporal. Este parâmetro referido indica os dias que faltam para se iniciar as estadas dos hóspedes em causa.

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:***



1. ***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.



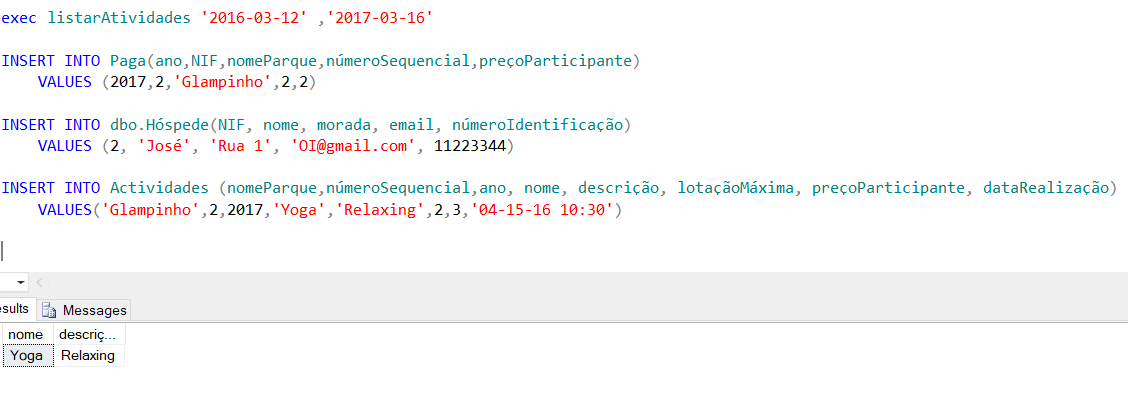
Por fim foi criado um procedimento armazenado que chama essa função.

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.

Uma imagem com captura de ecrã

Descrição gerada com confiança muito alta

***RESULTADO ESPERADO:***



1. ***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:

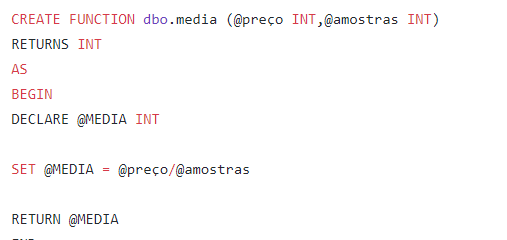
* Uma para ir guardando o valor total dos pagamentos selecionados;
* Outra para guardar o valor atual do pagamento da Fatura selecionada.
* Uma variável para ir guardando o número de amostras selecionadas.

Uma imagem com captura de ecrã

Descrição gerada com confiança muito alta

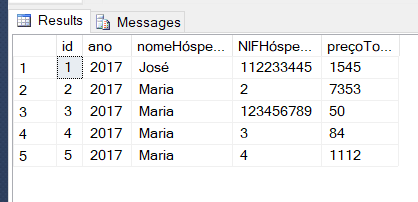
Uma vez que durantea execução deste procedimento não queremos que haja alterações à base de dados, a transacção tem o nível de isolamento SERIALIZABLE.

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

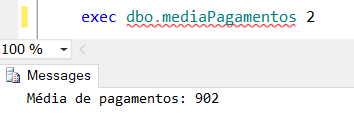


***RESULTADO ESPERADO:***

Faturas:



Média:



***(1545+50+1112)/3=902***

1. ***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.

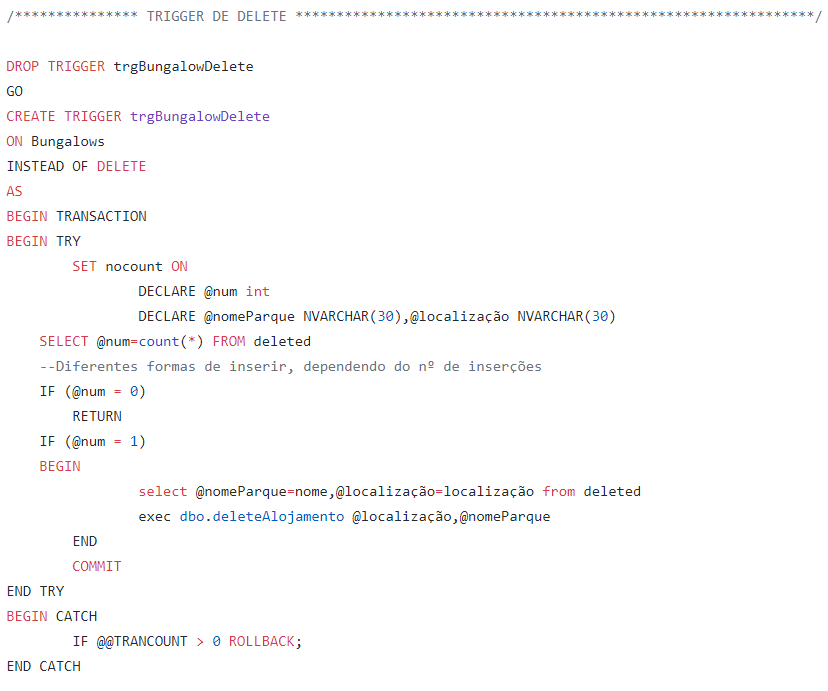
Ambos os triggers seguem a mesma lógica:

* 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 DELETE***



**TRIGGER DE UPDATE**

**Uma imagem com captura de ecrã

Descrição gerada com confiança muito alta**

***EXEMPLO:***

