ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Sistemas de Informação II

Turma LI52D | Inverno 2017/2018

Relatório 2ºTrabalho

Sistema de Informação II

Filipe Fé 42141 | Rui Lima 42200

Docente: Engenheiro Nuno Datia

Descrição

A segunda fase deste trabalho pode ser definida em duas partes distintas:

- Primeiramente a criação de uma aplicação ADO.NET que fosse capaz de fazer acesso a dados, neste caso mais concretamente à base de dados criada na primeira fase do trabalho.
- Alteração da aplicação referida anteriormente de forma a que use uma implementação de acesso a dados desenvolvida usando Entity Framework (EF).

Para a primeira parte desta fase do trabalho foi criada uma aplicação ADO.NET que se pode subdividir sucintamente em 3 partes:

1. Um conjunto de classes que são uma representação em .NET das entidades criadas na base de dados “Glampinho”;
2. Um conjunto de classes que implementam da Interface Mapper, tendo o objetivo de conseguir mapear cada uma das entidades instanciadas em .NET para a nossa base de dados, e vice-versa, mas também realizar operações que não estejam dependentes de uma entidade em específico, como por exemplo a classe ProcUtils.cs.

Relativamente ao ponto 2, todas as operações CRUD estão implementadas na classe AbstractMapper, por isso se alguma entidade não necessita de uma nova implementação nalguma dessas operações, é usada a implementação de AbstracMapper. Todas as outras operações que não estão relacionadas com nenhuma entidade especifica, mas que apenas se resumem a funcionalidades da aplicação, estão definidas numa classe denominada ProcUtils.cs.

Foram feitas algumas alterações ao trabalho realizado na primeira fase do projeto que serão descritas futuramente.

Aplicação ADO.NET

- ***Sem usar qualquer procedimento armazenado, obter o total pago por hóspede relativo a estadas num dado parque num intervalo de datas especificado.***

```
SELECT preçoTotal,idFactura FROM Factura INNER JOIN (
SELECT idFactura FROM Estada INNER JOIN(
SELECT A.id FROM HóspedeEstada INNER JOIN(
SELECT DISTINCT id FROM ParqueCampismo
INNER JOIN
AlojamentoEstada ON ParqueCampismo.nome=@parqueCampismo
)AS A ON HóspedeEstada.id=A.id and HóspedeEstada.NIF=@NIF )
AS B ON B.id=Estada.id and Estada.dataInício between @dataInicio and @dataFim)
AS C ON Factura.id = C.idFactura
```

  Para se calcular o total pago por um hóspede num determinado período de tempo é necessário ir buscar as estadas naquele intervalo de tempo num determinado parque que correspondem ao hóspede responsável em causa. Com essa informação já é possível encontrar o preço relativo a cada uma dessas estadas, através de uma junção de tabelas com a tabela Factura, onde se encontra o total pago pelo hóspede durante a sua estada.

- ***Sem usar qualquer procedimento armazenado, eliminar um dos parques (e respetivos alojamentos e estadas). Os dados sobre os hóspedes que não tenham estadas em qualquer outro parque também devem ser eliminados.***

  Para se poder apagar um Parque de Campismo é necessário apagar primeiramente tudo aquilo que está dependente dele, nomeadamente os seus alojamentos e estadas. Todas as entidades que tenham referências para o parque com determinado nome que é apagado, consequentemente também serão apagados (devido ao *delete on cascade* na chave estrangeira).

```
DELETE FROM ParqueCampismo WHERE nome=@nome
```

  Para se apagar as estadas, é necessário verificar quais os alojamentos daquele parque que estão associados às respetivas estadas.

```
SELECT A.id FROM Estada
INNER JOIN(
SELECT * FROM AlojamentoEstada WHERE nomeParque = @nomeParque)
AS A ON Estada.id = A.id
```

  Relativamente aos hóspedes, qualquer um que tenha uma estada no parque a eliminar e noutro qualquer não pode ser eliminado. Se apenas tiver uma estada no parque em causa, então é eliminado do sistema.

```
SELECT DISTINCT NIF FROM HóspedeEstada
EXCEPT
SELECT DISTINCT NIF FROM HóspedeEstada
INNER JOIN (
SELECT id FROM AlojamentoEstada WHERE nomeParque<>@nomeParque )
AS A ON A.id=HóspedeEstada.id
```

Entity Framework

A alteração da aplicação desenvolvida na primeira parte para uma implementação de acesso a dados desenvolvida usando Entity Framework (EF) não implicou grandes mudanças no nosso código anterior.

Foi usada a mesma aplicação nas duas implementações, diferindo apenas na sintaxe dos métodos que são chamados e nalgumas adaptações que são resultantes da forma como o código na EF está feito, portanto duma forma geral, a aplicação acaba por estar pouco dependente da forma como foi implementada a camada de acesso a dados.

A EF por si só já tem grande parte do código feito, nomeadamente a parte de acesso a dados e do seu armazenamento. Posto isto, podemos responder as perguntas teóricas colocadas no enunciado:

1. ***Comparar, da forma mais objetiva possível, as tecnologias EF e ADO.NET quanto à facilidade de programação e desempenho.***

   Relativamente à facilidade de programação, é notório a vantagem que a EF tem, visto que a maioria do código relativo a acesso e armazenamento de informação da nossa base de dados está feito, algo que na aplicação ADO.NET tinha de ser realizado pelo programador. Em contrapartida, isso implica que a EF tenha de carregar meta-informação para memória de forma a poder construir o modelo da nossa base de dados, o que demora tempo. No entanto, futuramente torna o desenvolvimento do nosso programa rápido e permite uma manutenção de grande parte do nosso código, algo que em ADO.NET seria mais complicado, visto que a responsabilidade desse código está do lado do programador. É possível concluir que a tecnologia EF torna a programação mais "confortável".

   Em termos de desempenho, é difícil fazer uma comparação muito precisa, visto que tudo depende da aplicação e também da base de dados em si. No entanto, se for uma aplicação pequena, o ADO.NET acaba por ser o mais adequado visto que tem um desempenho mais rápido. Caso seja uma aplicação de grande dimensão, é preferível perder desempenho utilizando a EF mas ter um código muito mais flexível.

2. ***Comparar as tecnologias EF e ADO.NET quanto à garantia da consistência dos dados.***

   Em termos de consistência de dados, a EF garante que depois de realizar uma operação os dados que estão na aplicação e na base de dados estão consistentes, pois quando é chamado o método SaveChanges() é feita essa verificação.

   Em ADO.NET é da responsabilidade do programador verificar se a transação não tornou os dados inconsistentes, portanto em termos de garantia, a EF acaba por levar vantagem.

Notas:

Juntamente com as duas aplicações serão enviados em anexo dois scripts sql, um para criar as base de dados, e outro para criar os procedimentos que serão usados pelas aplicações, bem como a criação à partida de um parque chamado Glampinho, que deve ser usado pelo utilizador de qualquer uma das aplicações para executar os vários comandos. Caso queira usar outro Parque, o mesmo deve ser criado em sql.

É da responsabilidade do utilizador garantir que os dados que insere estão correctos, uma vez que um erro no tipo do parâmetro ou tentativa de operar sobre um tuplo de qualquer tabela que não existe, pode não produzir os resultados esperados ou até mesmo lançar excepção.