Instituto Superior de Engenharia de Lisboa
Departamento de Engenharia de Eletrónica e Telecomunicações e de Computadores

Mestrado em Engenharia Informática e de Computadores
Arquitetura de Sistemas Distribuídos
2013/2014
Relatório Aula Prática Nº 3

Nome	Nº de Aluno	E-mail
Rui Miranda	A32342	a32342@alunos.isel.pt
David Coelho	A21359	a21359@alunos.isel.pt
Frederico Ferreira	A7066	a7066@alunos.isel.pt

DEETC, Dezembro de 2013

Exercício 1

1. Crie uma base de dados com as tabelas apresentadas no ficheiro Ex1.sql. Considere os exemplos de código fornecidos nas pastas Ex1.1 e Ex1.2, os quais são alterações do código apresentado na aula (Ex2 e Ex3).

a) Elimine todos os registos das duas tabelas e corra o código do exemplo Ex1.1. Verifique o comportamento.

b) Repita o mesmo processo, mas para o código Ex1.2.

Explique a razão porque os dois comportamentos que observa são diferentes.

Implementação

No âmbito destas duas alíneas, foram criados 8 ficheiros:

- `!Ex1.RunMe_AsAdmin.bat`
- `0.run.setupBD.bat`
- `0.run.setupBD.sql`
- `Ex1.clearAlunosEInteresses.bat`
- `Ex1.compilarEExecutarExec1.1.bat`
- `Ex1.compilarEExecutarExec1.2.bat`
- `Ex1.verificarAlunosEInteresses.bat`
- `Ex1.verificarAlunosEInteresses.sql`

O primeiro *batch* utiliza os restantes ficheiros para executar uma série de tarefas necessárias à execução destas duas alíneas, sendo a primeira a paragem do Coordenador de Transações Distribuídas ("MSDTC").

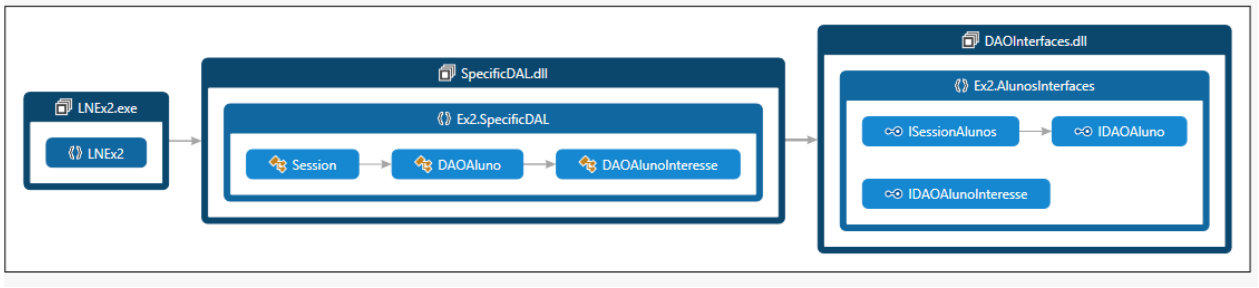
Alínea a)

O ficheiro `0.run.setupBD` é executado em seguida com o intuito de criar a BD e respetivas tabelas. O executável do exercício 1.1 é compilado e executado (com o auxilio de `Ex1.compilarE ExecutarExec1.1.bat`) sem qualquer problema:

- insere dois alunos e quatro interesses, os quais são visíveis quando se executa `Ex1.verificarAlunosEInteresses.bat`. A cada aluno foi adicionada a lista de interesses e a seguir foram inseridos os alunos na BD, um a seguir ao outro. Tudo decorreu dentro de uma única conexão e de uma transação definidas ao nivel da layer de negócio.

  A transação é controlada centralmente, correndo sobre uma conexão à base de dados, numa implementação elegante que permite a partilha de conexões e de transações, controladas ao nível da camada de negócio. As principais características desta implementação são:

  - uso da demarcação transacional dentro de uma conexão à base de dados (primeiro é criada a conexão e dentro dela é criada a transação);
  - existe uma partilha da conexão e da transação;
  - a classe DataAccessScope implementa a interface Idisposable, permitindo a sua utilização com o statement “using”. Cada instância controla se a conexão e a transação foram ou não criadas por sim, permitindo a partilha e re-uso da transação e da conexão.



Arquitetura de Implementação da Camada de Acesso a Dados.

Alínea b)

Depois de se apagarem estes dois alunos e respectivos interesses (recorrendo a `Ex1.clearAlunosEInteresses.bat`), o *batch* compila e executa o exercício 1.2 o qual falha por não estar disponível um Coordenador de Transações Distribuídas.

Este erro ocorre porque o exercício 1.2, mais que uma conexão à base de dados em simultâneo e logo a transação envolvida é promovida a transação distribuída.

Não estando o Coordenador de Transacções Distribuídas disponível para gerir esta transacção, a transacção falha, como se pode constatar pela mensagem de erro gerada: "`SqlException:MSDTC on server 'ALUNO_SIAD-PC\ONE' is unavailable.`"

O erro ocorre na abertura da segunda conexão, que é a relativa ao insert na tabela AlunosAssEst, dos interesses do primeiro aluno inserido (e cuja conexão também está aberta).

Exercício 2

2. Considere o código fornecido na pasta Ex2. Vá executando o código em single step e verificando nos pontos anotados o que observa na saída do no Sql Profiler (selects) e, em seguida faça "clear trace".

a) Justifique o que observa no Sql Profiler, tendo em conta os instantes em que as observações são realizadas.

b) Explique de forma resumida a razão do código TSQL observado.

Implementação

Ao executar o primeiro troço de código observar-se o seguinte:

EventClass	TextData	ApplicationName	NTUserName	LoginName	CPU	Reads	Writes
Audit Login	-- network protocol: LPC set quoted_identifier on...	EntityFramework	ALUNO_SIAD	ALUNO_...			
SQL:BatchStarting	SELECT [Extent1].[NumAl] AS [NumAl], [...	EntityFramework	ALUNO_SIAD	ALUNO_...			
SQL:BatchCompleted	SELECT [Extent1].[NumAl] AS [NumAl], [...	EntityFramework	ALUNO_SIAD	ALUNO_...	0	2	0
Audit Logout		EntityFramework	ALUNO_SIAD	ALUNO_...	0	2	0
RPC:Completed	exec sp_reset_connection	EntityFramework	ALUNO_SIAD	ALUNO_...	0	0	0
Audit Login	-- network protocol: LPC set quoted_identifier on...	EntityFramework	ALUNO_SIAD	ALUNO_...			
RPC:Completed	exec sp_executesql N'SELECT [Extent1].[NumAl...	EntityFramework	ALUNO_SIAD	ALUNO_...	0	17	0
Audit Logout		EntityFramework	ALUNO_SIAD	ALUNO_...	0	19	0
RPC:Completed	exec sp_reset_connection	EntityFramework	ALUNO_SIAD	ALUNO_...	0	0	0
Audit Login	-- network protocol: LPC set quoted_identifier on...	EntityFramework	ALUNO_SIAD	ALUNO_...			
RPC:Completed	exec sp_executesql N'SELECT [Extent1].[NumAl...	EntityFramework	ALUNO_SIAD	ALUNO_...	0	2	0

O primeiro evento assinalado, consiste na execução do seguinte código:

```
SELECT
    [Extent1].[NumAl] AS [NumAl],
    [Extent1].[Nome] AS [Nome]
    FROM [dbo].[Alunos] AS [Extent1]
```

Este select ocorre na primeira iteração do ciclo exterior (*foreach(var a in q) ...*)realizado sobre a colecção que resulta da expressão *Link*:

```
from a in ctx.Alunos select a
```

Os restantes dois eventos tem assinalados na figura consistem na selecção dos interesses de cada um dos dois alunos obtidos no select anterior. O segundo evento consiste na execução da seguinte *query,* (também a quando da iteração no ciclo *foreach(var i in a.AlunosAssEst)...*):

```
exec sp_executesql N'SELECT
    [Extent1].[NumAl] AS [NumAl],
    [Extent1].[nSeq] AS [nSeq],
    [Extent1].[Interesse] AS [Interesse]
    FROM [dbo].[AlunosAssEst] AS [Extent1]
    WHERE [Extent1].[NumAl] = @EntityKeyValue1'
, N'@EntityKeyValue1 int', @EntityKeyValue1=1111
```

O terceiro evento difere do segundo no valor atribuído a `@EntityKeyValue1` que no caso do terceiro evento é igual a 2222. Cada uma destas duas últimas queries são executadas quando se executa uma iteração do ciclo interior.

A segunda parte do programa consiste em repetir o código descrito anteriormente, mas agora para a seguinte expressão Link:

```
from a in ctx.Alunos.Include(a => a.AlunosAssEsts) select a
```

A expressão apesar de ser semelhante ao primeiro caso, implica uma diferença no comportamento assumido pela infraestrutura: quando o programa na primeira iteração do ciclo exterior acede à colecção de alunos, a seguinte query é executada na BD:

```
SELECT
    [Project1].[NumAl] AS [NumAl],
    [Project1].[Nome] AS [Nome],
    [Project1].[C1] AS [C1],
    [Project1].[NumAl1] AS [NumAl1],
    [Project1].[nSeq] AS [nSeq],
    [Project1].[Interesse] AS [Interesse]
    FROM ( SELECT
        [Extent1].[NumAl] AS [NumAl],
        [Extent1].[Nome] AS [Nome],
        [Extent2].[NumAl] AS [NumAl1],
        [Extent2].[nSeq] AS [nSeq],
        [Extent2].[Interesse] AS [Interesse],
        CASE WHEN ([Extent2].[NumAl] IS NULL) THEN CAST(NULL AS int) ELSE 1 END AS [C1]
        FROM  [dbo].[Alunos] AS [Extent1]
        LEFT OUTER JOIN [dbo].[AlunosAssEst] AS [Extent2]
             ON [Extent1].[NumAl] = [Extent2].[NumAl]
    )  AS [Project1]
    ORDER BY [Project1].[NumAl] ASC, [Project1].[C1] ASC
```

Trazendo para o cliente, num único select todos os dados dos Alunos e dos AlunosAssEst.

A partir deste ponto o programa nunca mais acede à BD, o que contrasta com o caso anterior. Por um lado o primeiro caso executa mais acessos à BD; por outro, o segundo caso sobrecarrega mais a memória do computador com informação que potencialmente pode ser desnecessária.

Neste exemplo, não verificámos qualquer diferença de comportamento com o uso e não uso de

```
ctx.Configuration.ProxyCreationEnabled = false;
```

ou com a linha comentada (o que significa o usar o valor default que é *true*).

Exercício 3

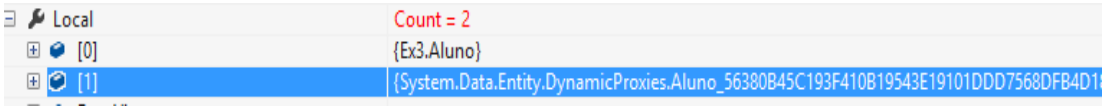
3. Considere o código c# fornecido na pasta Ex3. Vá executando o código em single step e seguindo as instruções no código. Anote as diferenças entre as duas vezes que executa o código (uma com e outra sem comentários no ponto 1).

Explique a razão de ser dos comportamentos que observa.

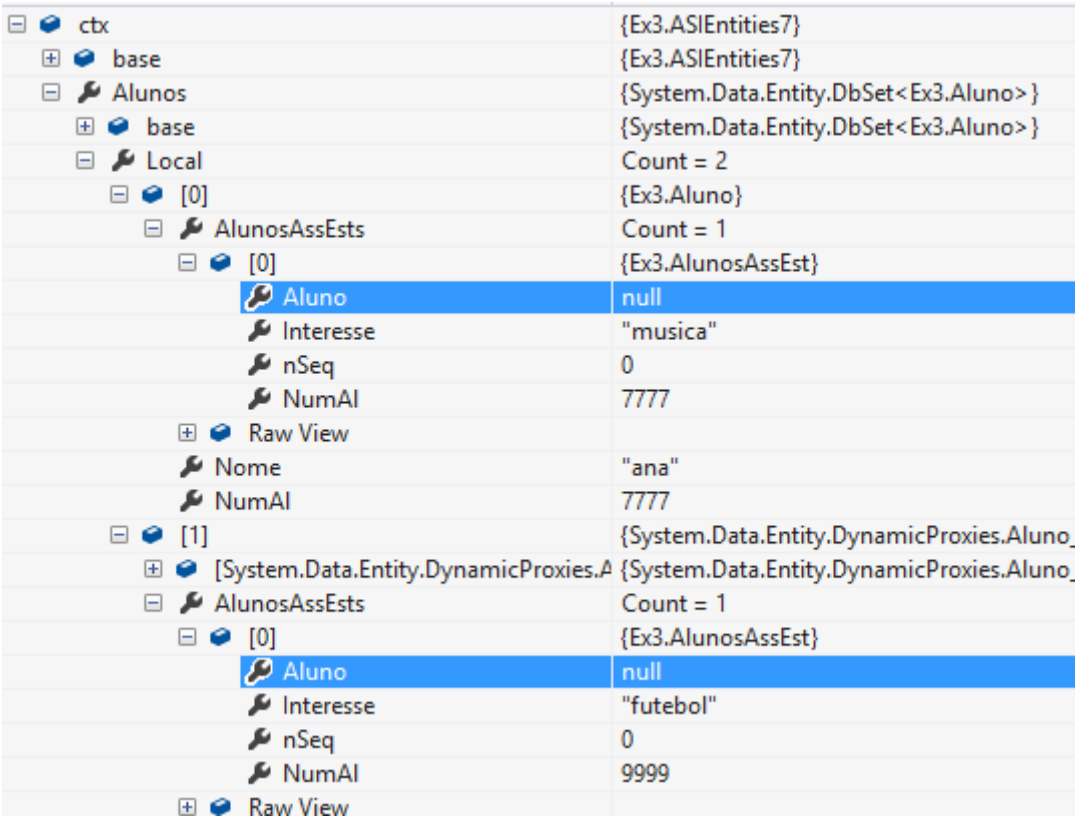
Implementação

Os dois alunos *al1* e *al2* foram criados usando mecanismos diferentes: shadow e proxy respectivamente.

O registo dentro de *Alunos.Local* do contexto *ctx* mostra essa diferença:



Como a *flag* de detecção de alterações está desligada, apenas os objectos criados através de *Context Create* são considerados na altura de persistir os objectos. Nesta situação, qualquer objecto criado através de *new* e adicionado através de *add* (o casso dos interesses com *new* AlunosAssEst) não é persistido.



Para persistir estes objectos é necessário indicar manualmente ao *Context* que foram associados novos objectos às instâncias criadas através de *Context Create*. É para isso que serve o `Context.ChangeTracker.DetectChanges`; indica que os dois alunos criados através do *Context* têm novas instâncias associadas que é preciso ter em conta na altura de persistir a informação dos alunos.

Sem se executar o `Context.ChangeTracker.DetectChanges` a *EntityFramework* limita-se a inserir na BD os dois alunos:

```
exec sp_executesql N'INSERT [dbo].[Alunos]([NumAl], [Nome])
VALUES (@0, @1)', N'@0 int, @1 varchar(60)', @0=7777, @1='ana'
```

```
exec sp_executesql N'INSERT [dbo].[Alunos]([NumAl], [Nome])
VALUES (@0, @1)', N'@0 int, @1 varchar(60)', @0=9999, @1='xavier'
```

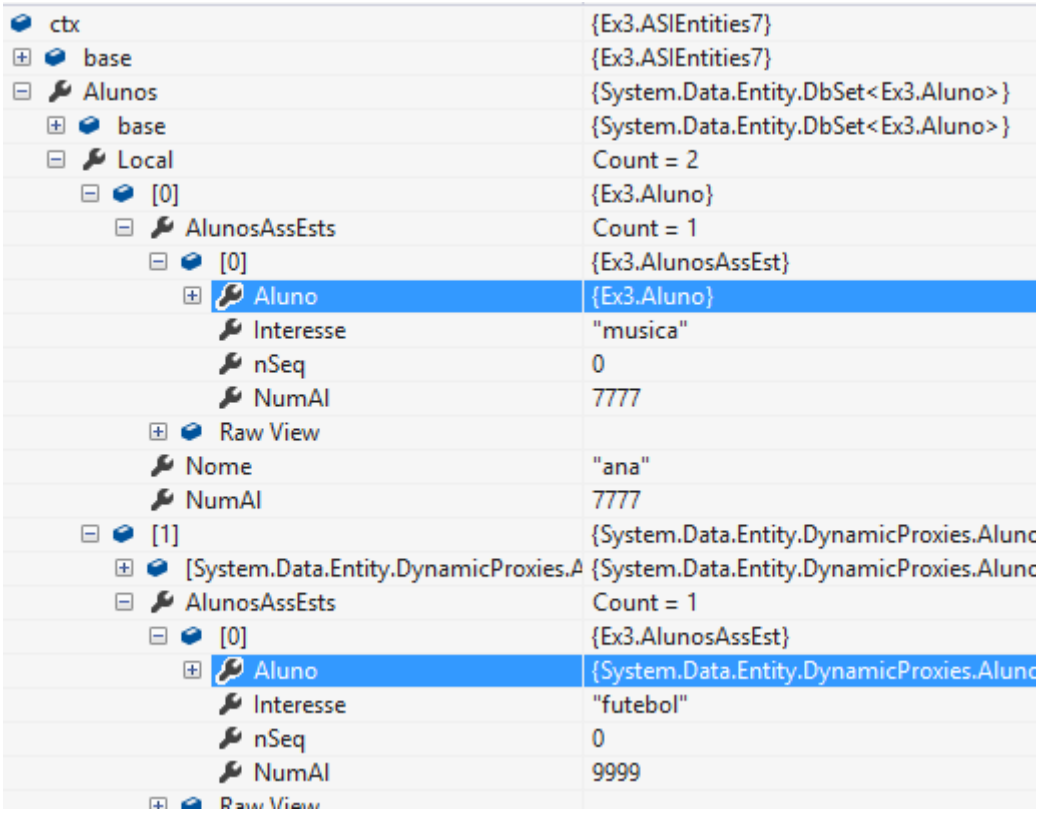
Invocando o `Context.ChangeTracker.DetectChanges`, a *EntityFramework* insere os dois alunos na BD, como se constatou anteriormente e em seguida insere os interesses instanciados através de `new`, adicionados a cada um dos alunos:

```
exec sp_executesql N'INSERT [dbo].[AlunosAssEst]([NumAl], [Interesse])
VALUES (@0, @1) SELECT [nSeq] FROM [dbo].[AlunosAssEst]
WHERE @@ROWCOUNT > 0 AND [NumAl] = @0 AND [nSeq] = scope_identity()'
, N'@0 int, @1 varchar(10)', @0=7777, @1='musica'

exec sp_executesql N'INSERT [dbo].[AlunosAssEst]([NumAl], [Interesse])
VALUES (@0, @1) SELECT [nSeq] FROM [dbo].[AlunosAssEst]
WHERE @@ROWCOUNT > 0 AND [NumAl] = @0 AND [nSeq] = scope_identity()'
, N'@0 int, @1 varchar(10)', @0=9999, @1='futebol'
```

Note-se que tudo isto se deve ao facto da *flag* de detecção de alterações estar desligada. Se assim não fosse (como se verifica em condições normais) o método `Context.ChangeTracker.DetectChanges` teria sido invocado sempre que se invocou o método *add* sobre as instâncias dos alunos.

A figura abaixo mostra o resultado depois da deteção das alterações com `Context.ChangeTracker.DetectChanges`, que actua nos 2 casos com e sem proxy:



Exercício 4

4. Na BD crie o aluno de nome "aaa" e número 1111 e outro de nome "bbbb" e número 4444. Execute o código fornecido na pasta Ex4, passo a passo e verifique o comportamento.

a) Explique porque observa esse comportamento.

b) Refaça o código de modo a que toda a execução dos contextos seja feita sem aberturas e fechos consecutivos das conexões e de forma a que não ocorra o comportamento que observou.

Implementação

No caso original são usados dois contextos diferentes, cada um com a sua conexão, ambos no âmbito da mesma transacção *(System.Transaction*). Cada um dos dois lê e altera um aluno diferente.

O primeiro contexto realiza a leitura do aluno com um comando semelhante ao seguinte:

```
SELECT TOP (1) [Extent1].[NumAl] AS [NumAl], [Extent1].[Nome] AS
[Nome]
    FROM [dbo].[Alunos] AS [Extent1]
    WHERE 1111 = [Extent1].[NumAl]
```

Depois de lido, o nome do aluno é alterado, um evento que não tem qualquer reflexo na BD.

É neste ponto -- antes do primeiro contexto realizar alterações ao aluno na BD -- que o segundo contexto é instanciado. Depois de abrir uma nova conexão para a BD, este novo contexto assume um comportamento idêntico ao do primeiro contexto, mas agora para um segundo aluno. Depois do nome do segundo aluno ser alterado, é invocado o método `SaveChanges` sobre o segundo contexto -- uma invocação que não apresenta qualquer consequência na BD. Depois de sair do *scope* do segundo contexto é também invocado o método `SaveChanges`, agora sobre o primeiro contexto, de forma a persistir as alterações ao nome do primeiro aluno. É neste ponto que a *EntityFramework* emite as instruções de *UPDATE* para actualizar o nome de ambos os alunos:

```
exec sp_executesql N'
UPDATE [dbo].[Alunos] SET [Nome] = @0
WHERE (([NumAl] = @1) AND ([Nome] = @2))',
N'@0 varchar(60), @1 int, @2 varchar(60)',
@0='xico', @1=1111, @2='aaa'


exec sp_executesql N'
UPDATE [dbo].[Alunos] SET [Nome] = @0
WHERE (([NumAl] = @1) AND ([Nome] = @2))',
N'@0 varchar(60), @1 int, @2 varchar(60)',
@0='xxxx', @1=4444, @2='bbbb'
```

Posteriormente resta apenas terminar a transacção com *commit*. Note-se que devido ao facto de se terem aberto duas conexões no âmbito desta transacção, esta passa a distribuida. Por este motivo, durante a execução deste programa o Coordenador de Transacções Distribuídas tem que estar activo.

O segundo exemplo, é uma versão modificada do primeiro exemplo. A modificação tem por objectivo evitar que a abertura do segundo contexto implique a abertura de uma nova conexão. Assim, em vez de se utilizar o construtor sem paramentos dos contextos -- algo que indica que essa instância de contexto deve criar a sua própria conexão para a base de dados -- é passada aos contextos uma conexão instanciada e controlada localmente.

No primeiro exemplo o seguinte código:

```
using (var ctx = new ASIEntities7())
```

é substituído por:

```
public partial class ASIEntities7 : DbContext {
     public ASIEntities7(EntityConnection cn) : base(cn,false) { }
}
(...)
EntityConnection cn = new EntityConnection("name=ASIEntities7");
using (var ctx = new ASIEntities7(cn)){
(...)
```

Exercício 5

5. Altere programa do exercício 2 para, usando instâncias de DBContext diferentes:

• Num contexto, inserir os alunos de nomes “Pedro” e “Paula” com, respectivamente os números 1001 e 2002.

• Noutro contexto, acrescentar os interesses “i1” e “i2” à Paula, usando os objecto do ponto anterior.

• Num terceiro contexto, eliminar a aluna Paula, usando os objectos dos pontos anteriores.

Implementação

O programa começa por criar uma conexão que será partilhada por todos os contextos e instância dois Alunos que representam "Pedro" e "Maria":

```
EntityConnection cn = new EntityConnection("name=ASIEntities7");

var alnPedro = new Aluno { NumAl = 1001, Nome = "Pedro" };
alnPedro.AlunosAssEsts = new HashSet<AlunosAssEst>();

var alnPaula = new Aluno { NumAl = 1002, Nome = "Paula" };
alnPaula.AlunosAssEsts = new HashSet<AlunosAssEst>();
```

Em seguida instância-se o primeiro contexto que persiste os dois alunos na BD:

```
using (var ctx1 = new ASIEntities7(cn))
{
    // em alternativa a usar o Sql Server Profiler, pode fazer:
    ctx1.Database.Log = Console.Write;

    ctx1.Alunos.Add(alnPedro);
    ctx1.Alunos.Add(alnPaula);

    ctx1.SaveChanges();
}
```

Note-se que por omissão, a detecção de alterações no contexto está activa, razão pela qual se dispensa a invocação ao método `DetectChanges` do contexto. O método `SaveChanges` provoca a execução do seguinte SQL:

```
exec sp_executesql N'INSERT [dbo].[Alunos]([NumAl], [Nome])
VALUES (@0, @1) ',
N'@0 int,@1 varchar(60)',@0=1001,@1='Pedro'
exec sp_executesql N'INSERT [dbo].[Alunos]([NumAl], [Nome])
VALUES (@0, @1)',
N'@0 int,@1 varchar(60)',@0=1002,@1='Paula'
```

O segundo contexto é criado com objectivo de se adicionar interesses à aluna "Paula":

```
using (var ctx2 = new ASIEntities7(cn))
{
    // em alternativa a usar o Sql Server Profiler, pode fazer:
    ctx2.Database.Log = Console.Write;

    var i1 = new AlunosAssEst { NumAl = 1002, Interesse = "i1" };
    var i2 = new AlunosAssEst { NumAl = 1002, Interesse = "i2" };

    alnPaula.AlunosAssEsts.Add(i1);
    alnPaula.AlunosAssEsts.Add(i2);
    ctx2.AlunosAssEsts.Add(i1);
    ctx2.AlunosAssEsts.Add(i2);

    ctx2.SaveChanges();
}
```

Note-se a adição manual dos dois interesses ao contexto. Este passo é necessário porque a adição dos interesses à aluna não é suficiente: o contexto não é informado sobre a adição destes objectos à aluna. Este é o preço a pagar pelo facto do objecto que representa a aluna ser um Simples Objecto de C# -- "POCO". O método `SaveChanges` provoca a execução do seguinte SQL:

```
exec sp_executesql N'INSERT [dbo].[AlunosAssEst]([NumAl],
[Interesse])
VALUES (@0, @1)
SELECT [nSeq] FROM [dbo].[AlunosAssEst]
WHERE @@ROWCOUNT > 0 AND [NumAl] = @0 AND [nSeq] =
scope_identity()',
N'@0 int,@1 varchar(10)', @0=1002, @1='i1'


exec sp_executesql N'INSERT [dbo].[AlunosAssEst]([NumAl],
[Interesse])
VALUES (@0, @1)
SELECT [nSeq] FROM [dbo].[AlunosAssEst]
WHERE @@ROWCOUNT > 0 AND [NumAl] = @0 AND [nSeq] =
scope_identity()',
N'@0 int,@1 varchar(10)', @0=1002, @1='i2'
```

Finalmente, o terceiro contexto é responsável pela remoção da aluna "Paula" da base de dados:

```
using (var ctx3 = new ASIEntities7(cn))
{
    // em alternativa a usar o Sql Server Profiler, pode fazer:
```

```
        ctx3.Database.Log = Console.Write;

        ctx3.Alunos.Attach(alnPaula);
        ctx3.Alunos.Remove(alnPaula);

        ctx3.SaveChanges();
    }
```

As únicas acções realizadas são a adição da aluna "Paula" ao contexto e a instrução de remoção. Esta última indica à *framework* que a aluna deve ser apagada da BD juntamente com as respectivas referencias, que neste caso se limitam a dois interesses:

```
exec sp_executesql N'DELETE [dbo].[AlunosAssEst]
WHERE (([NumAl] = @0) AND ([nSeq] = @1))',
N'@0 int,@1 int', @0=1002, @1=33
exec sp_executesql N'DELETE [dbo].[AlunosAssEst]
WHERE (([NumAl] = @0) AND ([nSeq] = @1))',
N'@0 int,@1 int', @0=1002, @1=34
exec sp_executesql N'DELETE [dbo].[Alunos]
WHERE (([NumAl] = @0) AND ([Nome] = @1))',
N'@0 int,@1 varchar(60)', @0=1002, @1='Paula'
```

Exercício 6

6. Altere o código do exercício 5 de forma a que a alteração dos alunos seja feita usando controlo de concorrência optimista.

Implementação

O exercício anterior foi alterado de forma a instanciar duas conexões: uma conexão da *EntityFramework* e uma conexão de *ADO.Net*. O objectivo é alterar o nome de um aluno concorrentemente através das duas tecnologias. Começa-se por inserir dois novos alunos, à semelhança do exercício anterior:

```
EntityConnection cn = new EntityConnection("name=ASIEntities7");
DbConnection dbcn = new SqlConnection( System.Configuration.
      ConfigurationManager.ConnectionStrings["basedados"].ConnectionString );

var alnPedro = new Aluno { NumAl = 1001, Nome = "Pedro" };
alnPedro.AlunosAssEsts = new HashSet<AlunosAssEst>();

var alnPaula = new Aluno { NumAl = 1002, Nome = "Paula" };
alnPaula.AlunosAssEsts = new HashSet<AlunosAssEst>();

using (var ctx1 = new ASIEntities7(cn)) {
    ctx1.Alunos.Add(alnPedro);
    ctx1.Alunos.Add(alnPaula);
    SaveContextChanges(ctx1, PoliticaConcorrencia.UsarInfoBD);
}
```

O método `SaveContextChanges` será analisado posteriormente, mas é essencialmente responsável pela invocação do método `SaveChanges` sobre o contexto `ctx1`.

Em seguida usa-se a *EntityFramework* para tentar alterar o nome de um aluno. O aluno é lido via *EntityFramework*, o seu nome é alterado, mas antes que se possa persistir essa alteração, o nome desse aluno é alterado na BD via *ADO.Net*. Ao tentar persistir as alterações realizadas sobre o aluno, a *EntityFramework* depara-se com um conflito, o qual se manifesta no âmbito do método `SaveContextChanges`. Como este método foi instruído para assumir uma política pessimista -- é válida a informação que for encontrada na BD -- resolve o conflito actualizando o aluno com o nome que foi encontrado na BD:

```
using (var ctxPessimista = new ASIEntities7(cn))
{
    Aluno alnPedroCtx = (from a in ctxPessimista.Alunos
                                where a.NumAl == 1001 select a).First();
    alnPedroCtx.Nome = "Pedro ContextoEF6";

    ExecuteDML(dbcn, "O ADO.Net vai mudar o nome do Pedro para 'Pedro ADO.Net'.",
                          "UPDATE [BD3_1].[dbo].[Alunos]
                         SET [Nome] = 'Pedro ADO.Net' WHERE [NumAl] = 1001");

    Console.WriteLine("O EF6 vai mudar o nome do Pedro para 'Pedro ContextoEF6'.
                           Antes de gravar, alnPedroCtx.Nome = {0}",
alnPedroCtx.Nome);
```

```
        SaveContextChanges(ctxPessimista, PoliticaConcorrencia.UsarInfoBD);
        Console.WriteLine("Depois de gravadas as alteracoes com uma politica
    pessimista
                          (vale o que esta na BD), o nome do Pedro no ambito do
    contexto
                          EF6 é: alnPedroCtx.Nome = {0})", alnPedroCtx.Nome);
    }
```

O exemplo final é muito semelhante ao anterior mas com a diferença de quando a *EntityFramework se* depara com o conflito no método `SaveContextChanges`, este assume uma politica optimista -- é válida a informação que está na entidade aluno -- e resolve o conflito actualizando o nome do aluno na BD com o nome que se encontra na entidade aluno no contexto *EntityFramework*:

```
using (var ctxOptimista = new ASIEntities7(cn))
{
    Aluno alnPedroCtx = (from a in ctxOptimista.Alunos
                         where a.NumAl == 1001 select a).First();
    alnPedroCtx.Nome = "Pedro ContextoEF6";

    ExecuteDML(dbcn, "O ADO.Net muda o nome para 'Pedro ADO.Net Outra Vez'.",
                "UPDATE [BD3_1].[dbo].[Alunos] SET [Nome] = 'Pedro ADO.Net Outra Vez'
                WHERE [NumAl] = 1001");

    Console.WriteLine("O EF6 vai mudar o nome do Pedro para 'Pedro ContextoEF6'.
                      Antes de gravar, alnPedroCtx.Nome = {0}", alnPedroCtx.Nome);
    SaveContextChanges(ctxOptimista, PoliticaConcorrencia.UsarInfoCtx);
    Console.WriteLine("Depois de gravadas as alteracoes com uma politica optimista
                       (vale o que esta no Contexto), o nome do Pedro no ambito do
                      contexto EF6 é: alnPedroCtx.Nome = {0})", alnPedroCtx.Nome);
}
```

O método `ExecuteDML` limita-se a executar o SQL que lhe é passado por parâmetro:

```
    private static void ExecuteDML(DbConnection cn, String msg, String cmdTxt) {
        Console.WriteLine(msg);
        DbCommand cmd = cn.CreateCommand();
        cmd.CommandText = cmdTxt;
        cn.Open(); cmd.ExecuteNonQuery();  cn.Close();
    }
```

O método `SaveContextChanges` grava as alterações realizadas no contexto invocando o método `SaveChanges` sobre o mesmo. Mais; invoca o método as vezes que forem necessárias até que não ocorra uma excepção nessa invocação.

Quando ocorre uma excepção que indica um conflito, verifica-se a politica que foi escolhida por quem invocou `SaveContextChanges`: se se escolheu uma politica pessimista, a entidade que provocou o conflito é simplesmente actualizada com os novos dados que se encontram na BD, perdendo-se assim os dados que se encontravam na entidade. Se se escolheu uma politica optimista, a entidade que provocou o conflito é alterada de forma a que os seus valores **originais** batam certo com os novos dados que se encontram na BD. Desta forma, na próxima invocação do método `SaveChanges`, não irá ocorrer um conflito e perdem-se assim os dados que se encontravam na BD.

```
private enum PoliticaConcorrencia { UsarInfoBD, UsarInfoCtx };
private static void SaveContextChanges(DbContext ctx,
      PoliticaConcorrencia politicaConcorrencia = PoliticaConcorrencia.UsarInfoCtx) {
    bool falha;
    do {
        falha = false;
        try { ctx.SaveChanges(); }
        catch (DbUpdateConcurrencyException ex) {
            Console.WriteLine("Ocorreu uma DbUpdateConcurrencyException
                                durante o SaveChanges.");
            falha = true;
            switch (politicaConcorrencia) {
                  case PoliticaConcorrencia.UsarInfoBD: {
                      // ignorar as alterações feitas no contexto e usar a
                        // informação corrente na BD (estado = unchanged)
                      ex.Entries.Single().Reload();
                      break;
                  }
                    case PoliticaConcorrencia.UsarInfoCtx: {
                          // esmagar as alterações na BD
                          var entry = ex.Entries.Single();
                          var dbValues = entry.GetDatabaseValues();
                          entry.OriginalValues.SetValues(dbValues);
                          break;
                  }
                    default:
                       throw new Exception(
                           "Politica desconhecida: " + politicaConcorrencia.ToString());
            }
        }
    } while (falha);
}
```