

SISTEMAS DE INFORMAÇÃO 2

Trabalho Prático

***Dedicatória:***

*Às famílias dos elementos deste grupo pelo apoio e pelo esforço gigante com que todos os dias nos acompanham e nos reforçam positivamente. Simples palavras não permitem expressar o agradecimento que se impõe.*

**Índice Remissivo:**

1. Extensões e restrições aos dados iniciais, 6

2 - Código T-SQL, 7

2.a - Inserir uma nova empresa ou área de intervenção, 7

2.b - Actualizar os dados de uma empresa, 7

2.c - Reportar uma ocorrência, 8

2.d - Cancelar uma ocorrencia, 9

2.e - Dar inicio à resolução de uma ocorrencia, 9

2.f - Assinalar a finalização da prestação de serviço numa área de intervenção de uma dada ocorrência, 10

2.g - Apresentar, para cada empresa, o número total de ocorrências organizadas por tipo, 11

2.h - Listar as ocorrências em situação de incumprimento face ao prazo estabelecido para a sua resolução, 11

2.i - Indicar, para uma determinada área de intervenção, qual a empresa com maior número de ocorrências do tipo “crítico” que reportou ocorrências nessa área, 11

2.j - Listar os funcionários que nunca tenham tido a coordenação de uma ocorrência do tipo “crítico”, 12

2.k - Criar um processo que permita registar pontos de crédito, 12

3 - Crie uma aplicação .NET que, usando para acesso a dados o (ADO.NET) e Entity Framework, permita disponibilizar as seguintes funcionalidades, 13

**Enunciado:**

Uma empresa, com actividade na área da assistência técnica, pretende agilizar o seu negócio adoptando um sistema de informação para gestão de ocorrências. No sistema deve ser mantido registo das empresas clientes e respectivas ocorrências classificadas por áreas de intervenção.

As empresas cliente são caracterizadas pelo número de identificação de pessoa colectiva (NIPC), pela designação comercial e morada fiscal. Uma empresa possui instalações geograficamente distribuídas, onde é prestada assistência técnica. As instalações são identificadas por um código numérico, têm uma localização geográfica (coordenadas e morada) e opcionalmente uma descrição. Cada empresa estabelece, para cada uma das suas instalações, um conjunto de sectores onde é prestada assistência técnica. Os sectores são identificados de acordo com a sintaxe: **S<piso><zona>** (*e.g. S1A, S2A, S0B, …*). Cada sector tem informação descritiva e informação acerca da existência, ou não, de extintor para combate a incêndios.

As áreas de intervenção, no âmbito das quais é prestada assistência técnica, são identificadas por um código alfanumérico gerado automaticamente, uma designação (e.g. “Assistência Técnica”) e uma descrição. Cada área de intervenção tem um conjunto de funcionários do qual um subconjunto é de coordenadores. Os funcionários são identificados por um número e têm nome, idade e data de obtenção da habilitação para desempenho de funções na área de intervenção onde trabalham. As empresas cliente reportam no sistema de informação as suas ocorrências. Uma ocorrência tem um identificador numérico, data e hora de entrada, estado, data e hora da última alteração de estado e um tipo (“urgente”, “crítico” ou “trivial”)1. Para cada ocorrência é indicado o sector da instalação onde é requerida a intervenção e podem ser indicadas no máximo três áreas de intervenção.

Uma ocorrência tem os seguintes estados: inicial, em processamento, em resolução, recusado, cancelado ou concluído. O estado inicial é atribuído quando a ocorrência é reportada. O estado recusado significa que a ocorrência está fora do âmbito do contrato. O estado cancelado 2, indica que já não é necessária intervenção. O estado em processamento denota que foi aceite a ocorrência. O estado em resolução indica que foram atribuídos funcionários para coordenarem cada uma das áreas de intervenção associadas à ocorrência. Para cada área de intervenção da ocorrência, é atribuído um coordenador (a responsabilidade é atribuída automaticamente ao funcionário que tiver menos ocorrências em resolução). O estado de uma ocorrência transita para concluído logo que estejam terminados os trabalhos em todas as áreas de intervenção.

**Introdução:**

O trabalho final da cadeira de SI2, documentado neste documento, tem como objectivo a aplicação dos conceitos avançados da linguagem SQL incluindo processamento transaccional envolvendo concorrência, procedimentos armazenados e gatilhos e utilização da API de acesso a dados ADO.NET (incluindo Entity Framework).

O caso apresentado no enunciado é o de uma empresa prestadora de serviços de assistência técnica. A operação desta empresa centra-se na gestão de ocorrências, levantadas por empresas clientes que possuem actividade dispersa por diversos locais de actuação, identificados por sectores perfeitamente definidos e identificados pelos pisos e zonas das respectivas instalações onde operam.

Estas ocorrências são caracterizadas quanto à unidade de actuação em que ocorrem, classificadas quanto ao tipo (“**critico**”, “**urgente**” ou “**trivial**”) e acompanhadas com recurso a sinalizadores de estado, estando previstos os estados “**inicial**”, “**em** **processamento**”, “**em** **resolução**”, “**recusado**”, “**cancelado**” ou “**concluído**”.

Dos requisitos entendemos ainda que deve ser prevista a possibilidade de afectação de meios humanos ao tratamento das ocorrências. Neste particular, deve ser observada a área de competência para intervenção dos funcionários, e a classificação entre funcionários que reúnem competência para coordenar uma determinada intervenção.

Da análise ao enunciado e SQL fornecido para geração de base de dados resulta o modelo **EA**[[1]](#footnote-1) cujo diagrama apresentamos de seguida:

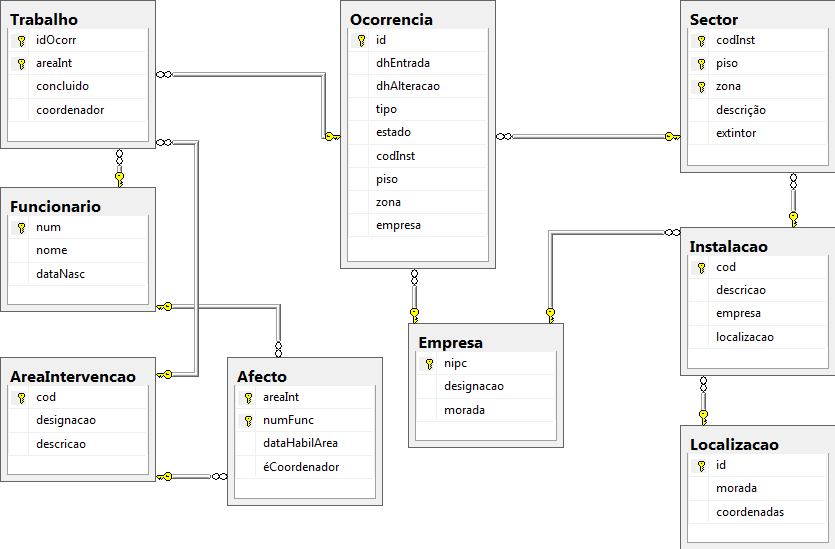


Ilustração 1 - Diagrama de entidades e respectivas associações – BD SQL Server

Avançaremos para a resolução dos trabalhos propostos:

1. Tendo por base o modelo físico fornecido, acrescente-lhe as restrições e as entidades e associações que achar convenientes para suportar a implementação das funcionalidades pedidas. Não devem ser feitas alterações de estrutura às tabelas e associações já existentes sem a prévia consulta ao docente:

Da análise que fizemos ao modelo, e em observação dos requisitos presentes no enunciado resultou a introdução de um conjunto de restrições e alterações ao modelo fornecido que no nosso entendimento permite responde de forma adequada ao solicitado.

Nas páginas que se seguem apresentamos as alterações introduzidas ao modelo, por entidade:

**Ocorrência:**

Pré-definimos o valor por defeito para alguns campos da tabela de ocorrências. O estado para novas ocorrências passa a “***inicial***” por defeito, e os campos para data de Entrada e alteração passam a ser por defeito preenchidos com a data e hora da inserção.

Adicionámos restrições quanto às inserções válidas para os campos “***estado***” e “***tipo***” em linha com os valores determinados no enunciado.

ALTER TABLE [dbo].[Ocorrencia] ADD DEFAULT ('inicial') FOR [estado]

ALTER TABLE [dbo].[Ocorrencia] ADD DEFAULT GETDATE() FOR dhEntrada

ALTER TABLE [dbo].[Ocorrencia] ADD DEFAULT GETDATE() FOR dhAlteracao

ALTER TABLE [dbo].[Ocorrencia]

WITH CHECK ADD CHECK

(([estado]='concluído' OR [estado]='cancelado' OR [estado]='recusado' OR [estado]='em resolução' OR [estado]='em processamento' OR [estado]='inicial'))

ALTER TABLE [dbo].[Ocorrencia]

WITH CHECK ADD CHECK

(([tipo]='trivial' OR [tipo]='crítico' OR [tipo]='urgente'))

**Funcionario:**

Estabelecemos o valor por omissão para o campo ***num***. Definimos a ***SEQUENCE dbo.FuncionarioCod***, a que recorremos com ***NEXT VALUE FOR*** para obtenção do valor seguinte para o campo

CREATE SEQUENCE dbo.FuncionarioCod AS int START WITH 1 INCREMENT BY 1

CREATE TABLE Funcionario(

num int primary key DEFAULT (NEXT VALUE FOR FuncionarioCod),

nome varchar(250) not null,

dataNasc date not null

)

**Instalação:**

Semelhante. Definição de sequência para o campo ***cod***:

CREATE SEQUENCE dbo.InstalacaoCod AS int START WITH 1 INCREMENT BY 1

create table Instalacao(

cod int primary key DEFAULT (NEXT VALUE FOR InstalacaoCod),

descricao varchar (150) not null,

empresa int references Empresa not null,

localizacao int references Localizacao not null

)

1. Apresente o código T-SQL (e respectivas extensões, vistas, funções, procedimentos armazenados e gatilhos) que, para além de outras operações que queira implementar, permita implementar as funcionalidades abaixo apresentadas, garantindo os requisitos de negócio enunciados anteriormente.
2. Inserir uma nova empresa ou área de intervenção:

Optamos por criar procedimentos armazenados para reunir o conjunto de operações que venham a ser necessárias à execução das tarefas solicitadas. Apresentamos também os respectivos exemplos de utilização.

As operações que se seguem não requerem alteração ao nível de isolamento, uma vez que reúnem operações de escrita simples sem leituras pelo que são bem formadas de duas fases (mantém o **LOCK**[[2]](#footnote-2) até ao final da transacção) e portanto, seguras do ponto de vista de acesso concorrente.

Na documentação do fabricante podemos ler o seguinte no que diz respeito ao comportamento previsto para operações de escrita:

*“An UPDATE statement always acquires an exclusive (X) lock on the table it modifies,*

*\* and holds that lock until the transaction completes. With an exclusive (X) lock,*

*\* no other transactions can modify data.”*

Citação 1 – Retirada de msdn.microsoft.com

Inserir uma empresa:

CREATE PROCEDURE InserirEmpresa @nipc int, @designacao varchar(50), @morada varchar(255) AS

BEGIN

INSERT INTO Empresa VALUES(@nipc, @designacao, @morada)

END

GO

Exemplo de utilização:

EXEC InserirEmpresa 600016234, 'Instituto Superior de Engenharia de Lisboa', 'Rua Conselheiro Emídio Navarro 1, 1959-007 Lisboa'

Inserir uma Área de Intervenção:

CREATE PROCEDURE InserirAreaIntervencao @id int OUT, @designacao varchar(50), @descricao varchar(250) = null AS

BEGIN

INSERT INTO AreaIntervencao VALUES(@designacao, @descricao)

SET @id = SCOPE\_IDENTITY()

END

GO

Exemplo de utilização:

EXEC InserirAreaIntervencao @AreaID OUT, 'Assistência Técnica'

1. Actualizar os dados de uma empresa:

As operações de escrita são bem formadas de 2 fases, como tal, como referido no ponto anterior, não é necessário especificar um nível de isolamento para esta transacção:

CREATE PROCEDURE EditarEmpresa @nipc int, @designacao varchar(50), @morada varchar(255) AS

BEGIN

UPDATE Empresa SET designacao=@designacao, morada=@morada WHERE nipc=@nipc

END

GO

1. Reportar uma ocorrência:

Neste caso criamos um procedimento armazenado que reúne todas as operações relacionadas a esta tarefa.

Especificamos um nível de isolamento **REPEATABLE READ**[[3]](#footnote-3) para esta transacção conferindo a necessária protecção contra eventuais erros dos seguintes tipos:

- **DIRTY READ** (que podem ocorrer em transacções não isoladas, ou com nível **READ** **UNCOMMITTED**[[4]](#footnote-4), o que basicamente permite ler registos que possuam lock para escrita, ou seja, que estão em actualização por uma outra transacção);

- **NON REPEATABLE READ** (que podem ocorrer em transacções com nível de isolamento **READ COMMITTED**[[5]](#footnote-5), o que bloqueia a leitura a registos que possuam lock para escrita, mas permite que outras transacções alterem dados já lidos no âmbito desta transacção):

CREATE PROCEDURE ReportarOcorrencia @tipo char(7), @empresa int, @codInst int, @piso int, @zona char(1), @id int OUT AS

BEGIN

SET XACT\_ABORT ON

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

BEGIN TRANSACTION

BEGIN TRY

/\* RI: empresa tem que corresponder à empresa da instalacao da ocorrencia \*/

SELECT Empresa.nipc from Instalacao

INNER JOIN Empresa ON Empresa.nipc = Instalacao.empresa

INNER JOIN Sector ON Sector.codInst = Instalacao.cod

WHERE Empresa.nipc = @empresa AND Instalacao.cod=@codInst AND piso=@piso AND zona=@zona

IF @@ROWCOUNT > 0

BEGIN

INSERT INTO Ocorrencia (tipo, empresa, codInst, piso, zona) VALUES (@tipo, @empresa, @codInst, @piso, @zona)

SET @id = SCOPE\_IDENTITY()

END

COMMIT

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

ROLLBACK

END CATCH

END

GO

Definimos um gatilho que responde à acção de actualização sobre a tabela de **Ocorrencias**, actualizando o campo “***dhAlteracao***” a cada actualização.

CREATE PROCEDURE EditarEmpresa @nipc int, @designacao varchar(50), @morada varchar(255) AS

BEGIN

UPDATE Empresa SET designacao=@designacao, morada=@morada WHERE nipc=@nipc

END

GO

1. Cancelar uma ocorrência:

Pelos motivos já referidos anteriormente, não há necessidade de especificar um nível de isolamento específico para esta transacção. Especificamos uma restrição: No nosso entendimento não faz sentido que uma empresa cliente possa cancelar uma ocorrência numa fase em que já existam recursos materiais e humanos afectados à respectiva resolução. Como tal, o cancelamento é restrito a ocorrências cujo estado seja “***inicial***” ou “***em*** ***processamento***”:

CREATE PROCEDURE CancelarOcorrencia @id int AS

BEGIN

UPDATE Ocorrencia SET estado='cancelado' WHERE id=@id AND (estado='inicial' OR estado='em processamento')

END

GO

1. Dar início à resolução de uma ocorrência em processamento:

Criamos uma função auxiliar que devolve o número do **Funcionário**, que seja **coordenador**, com o menor número de Ocorrências Activas e que portanto está apto a coordenar a resolução da presente ocorrência:

CREATE FUNCTION dbo.FuncionarioComMenosOcorrenciasActivas(@areaID INT) RETURNS INT AS

BEGIN

DECLARE @num INT

SELECT TOP(1) @num=num FROM Funcionario

INNER JOIN Afecto ON Afecto.numFunc=Funcionario.num AND Afecto.éCoordenador=1 AND Afecto.areaInt=@areaID

INNER JOIN AreaIntervencao ON AreaIntervencao.cod=Afecto.numFunc

LEFT OUTER JOIN (

SELECT coordenador, COUNT(coordenador) as cnt FROM Trabalho

INNER JOIN

(SELECT id FROM Ocorrencia WHERE estado='em resolução') AS Ocorr ON Trabalho.idOcorr=Ocorr.id

GROUP BY coordenador) as Afc ON Afc.coordenador=Funcionario.num

ORDER BY Afc.cnt ASC

RETURN @num

END

GO

Procedimento que realiza a atribuição de uma ocorrência usando a lógica de negócio prevista (recorre à função descrita acima para determinar o funcionário que ficará encarregue da resolução). Este procedimento recebe como parâmetro um inteiro que é o identificador da ocorrência:

CREATE PROCEDURE AssignarOcorrencia @id int AS

BEGIN

SET XACT\_ABORT ON

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

BEGIN TRANSACTION

BEGIN TRY

DECLARE @emProcessamento int

DECLARE @funcionario int

SELECT @emProcessamento=count(\*) FROM Ocorrencia WHERE id=@id AND estado='em processamento'

IF @emProcessamento > 0

BEGIN

BEGIN

UPDATE Trabalho SET coordenador = dbo.FuncionarioComMenosOcorrenciasActivas(areaInt) WHERE Trabalho.idOcorr=@id

UPDATE Ocorrencia SET estado='em resolução' WHERE id=@id

END

END

COMMIT

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

ROLLBACK

END CATCH

END

GO

1. Assinalar a finalização da prestação de serviço numa área de intervenção de uma dada ocorrência:

O estado de uma ocorrência transita para concluído logo que estejam terminados os trabalhos em todas as áreas de intervenção. Pretendemos portanto assinalar a conclusão de prestação de serviço para uma área de intervenção, o que significa criar um **Trigger** para alterações ao campo "**concluído**" da tabela "**Trabalho**". A acção despoletada passa por verificar se TODOS os registos da tabela “Trabalho” que estão relacionados com a presente ocorrência se encontram no estado "concluído", e se for esse o caso, actualiza a tabela "**Ocorrencia**", alterando o estado para "concluído".

Criamos uma função auxiliar que obtem o número de Trabalhos em curso para uma determinada ocorrência (passada por parâmetro):

CREATE FUNCTION numTrabalhosEmCursoPorOcorrencia(@idOcorr int) RETURNS int

AS

BEGIN

RETURN (SELECT COUNT(\*) FROM Trabalho WHERE idOcorr = @idOcorr AND concluido = 0)

END

GO

Criamos o **Trigger** para updates à tabela **Trabalho**. Aumentamos o nível de isolamento para **SERIALIZABLE**[[6]](#footnote-6), por forma a poder ter um **lock ao critério**[[7]](#footnote-7) na função “***numTrabalhosEmCursoPorOcorrencia***” e assim evitar a ocorrência de erros **PHANTOM** **READ**[[8]](#footnote-8):

CREATE TRIGGER conclusãoTrabalhoAreaIntervencao ON Trabalho AFTER UPDATE

AS

BEGIN

SET XACT\_ABORT ON

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

BEGIN TRANSACTION

BEGIN TRY

DECLARE @idOcorr int, @numTrabalhosEmCurso int, @concluido bit

SET @idOcorr = (SELECT idOcorr from inserted)

SET @numTrabalhosEmCurso = dbo.numTrabalhosEmCursoPorOcorrencia(@idOcorr)

IF (UPDATE(concluido) AND @numTrabalhosEmCurso = 0)

UPDATE Ocorrencia SET estado = 'concluído' WHERE id = @idOcorr

COMMIT

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0

ROLLBACK

END CATCH

END

GO

1. Apresentar, para cada empresa, o número total de ocorrências organizadas por tipo:

Criamos uma **View** [[9]](#footnote-9) que apresenta o total de ocorrências por tipo:

CREATE VIEW TotalOcorrenciaPorEmpresaPorTipo AS

SELECT oc.empresa, oc.tipo, COUNT(oc.tipo) AS nrOcorrTipo FROM Ocorrencia AS oc

INNER JOIN Empresa e

ON oc.empresa = e.nipc

GROUP BY oc.tipo, oc.empresa

GO

1. Listar as ocorrências em situação de incumprimento face ao prazo estabelecido para a sua resolução:

Criamos uma **View** para apresentação do resultado pedido:

CREATE VIEW OcorrenciasEmIncumprimento AS

SELECT \*, (DATEDIFF(HOUR, oc.dhEntrada, oc.dhAlteracao)) AS HOURa FROM Ocorrencia oc

WHERE ((oc.tipo = 'urgente' AND DATEDIFF(HOUR, oc.dhEntrada, oc.dhAlteracao) > 48) OR

(oc.tipo = 'crítico' AND DATEDIFF(HOUR, oc.dhEntrada, oc.dhAlteracao) > 12)) AND

(oc.estado NOT IN ('recusado', 'cancelado', 'concluído'))

GO

1. Indicar, para uma determinada área de intervenção, qual a empresa com maior número de ocorrências do tipo “crítico” que reportou ocorrências nessa área:

Criamos uma **Função**[[10]](#footnote-10) não escalar que recebe o identificador da área de intervenção e retorna uma tabela com o identificador da empresa com o maior numero de ocorrências criticas e a respectiva contagem de ocorrências críticas.

CREATE FUNCTION dbo.ObterEmpresaComMaxCritoOcorrPorAreaInt(@areaInt int)

RETURNS @maxTable TABLE

(

empresa int PRIMARY KEY NOT NULL,

Ocurrencias int

) AS

BEGIN

INSERT INTO @maxTable

SELECT empresa, COUNT(empresa) as cnt FROM Trabalho INNER JOIN Ocorrencia ON Ocorrencia.id = Trabalho.idOcorr AND Ocorrencia.tipo='crítico'

WHERE Trabalho.areaInt=@areaInt GROUP BY empresa

HAVING COUNT(empresa) =

(SELECT TOP(1) MAX(cnt) as maxCnt FROM

(SELECT empresa, COUNT(empresa) as cnt FROM Trabalho INNER JOIN Ocorrencia ON Ocorrencia.id = Trabalho.idOcorr AND Ocorrencia.tipo='crítico'

WHERE Trabalho.areaInt=@areaInt GROUP BY empresa) as x2 GROUP BY empresa)

RETURN

END

GO

1. Listar os funcionários que nunca tenham tido a coordenação de uma ocorrência do tipo “crítico”:

Para esta tarefa criamos um **procedimento armazenado**[[11]](#footnote-11) que lista os funcionários que correspondem ao critério. Esta transacção não requer isolamento, uma vez que se trata de uma única operação de leitura, não havendo necessidade de aumentar o nível de isolamento, sob pena de penalizar o comportamento do motor de base de dados:

CREATE PROCEDURE ListarFuncionariosCoordenadoresSemOcorrenciasCriticas AS

BEGIN

DECLARE @ocorrenciaTipo as varchar(7)

SET @ocorrenciaTipo = 'crítico'

SELECT fu.nome FROM Funcionario fu

LEFT JOIN

(SELECT f.num, f.nome, oc.tipo, COUNT(oc.tipo) AS NrOcorrencias FROM Ocorrencia oc

INNER JOIN Trabalho t

ON t.idOcorr = oc.id AND oc.tipo = @ocorrenciaTipo

INNER JOIN Funcionario f

ON f.num = t.coordenador

GROUP BY f.num, f.nome, oc.tipo

HAVING COUNT(oc.tipo) > 0

) AS Oco

ON oco.num = fu.num

WHERE oco.NrOcorrencias IS NULL

END

GO

1. Criar um processo que permita em determinado momento no tempo, para todas as ocorrências que estejam em incumprimento, registar um ponto (de crédito) a cada empresa visada. Estes pontos serão um dia mais tarde convertidos em cheques-bónus (fora do âmbito deste projecto). Para tal deve:

i. Criar as tabelas auxiliares necessárias ao registo de pontos por empresa;

ii. Garantir que, uma empresa não recebe vários pontos pela mesma ocorrência, mesmo que o

processo se execute várias vezes com a mesma ocorrência ainda em incumprimento;

iii. Manter registo de todas as ocorrências que deram origem a pontos, e a sua data de

processamento;

Neste ponto, optámos por criar um procedimento armazenado que verifica a existência da tabela auxiliar para registo de incumprimentos e cria a mesma se necessário, procedendo ao registo do incumprimento, desde que verificadas as condições previstas no enunciado, nomeadamente garantir que uma empresa não recebe vários pontos pela mesmo empresa, mesmo que o processo se execute varias vezes com a mesma ocorrência. Este procedimento faz uso da View criada na alínea 2.h:

CREATE PROCEDURE processarPontosPorIncumprimento AS

BEGIN

IF(OBJECT\_ID('registoIncumprimento') is null)

CREATE TABLE registoIncumprimento(

idOcorr int references Ocorrencia,

dhProcessamento datetime not null DEFAULT GETDATE()

)

INSERT INTO registoIncumprimento(idOcorr)

SELECT id FROM OcorrenciasEmIncumprimento WHERE id NOT IN (SELECT idOcorr FROM registoIncumprimento)

END

GO

1. Crie uma aplicação .NET que, usando para acesso a dados o (ADO.NET) e Entity Framework, permita disponibilizar as seguintes funcionalidades:
2. Admita que se pretende que seja possível carregar no sistema um conjunto de registos de ocorrências (e respectivo detalhe das áreas de intervenção) através de um documento xml criado para o efeito, evitando assim a inserção manual de cada ocorrência.
3. Crie um schema xml (XSD) que permita validar a estrutura de um documento xml que contenha informação sobre o registo de um conjunto de ocorrências, com o seu respectivo detalhe
4. Apresente exemplos de documentos xml para carregamento:
5. Crie uma aplicação (ou integre na aplicação realizada na alínea anterior) a funcionalidade de carregar o ficheiro xml, inserindo a respectiva informação na base de dados:

1. **EA** ou **Entidade Associação**: Modelo conceptual representativo dos dados do ponto de vista do programador. Na ilustração 1 apresentamos o diagrama gerado pelo SqlServer Management Studio a partir do modelo fornecido. [↑](#footnote-ref-1)
2. **LOCK** –Protecção aplicada contra escrita ou leitura. [↑](#footnote-ref-2)
3. REPEATABLE READ: Nível de isolamento. Especifica que as instruções não podem ler dados que foram modificados, mas que ainda não foram confirmados por outras transações e que nenhuma outra transação pode modificar dados que foram lidos pela transação atual até que a transação atual seja concluída *(in http://msdn.microsoft.com/)* [↑](#footnote-ref-3)
4. READ UNCOMMITTED: Nível de isolamento. Especifica que as instruções podem ler linhas que foram modificadas por outras transações, mas que ainda não foram confirmadas *(in http://msdn.microsoft.com/)* [↑](#footnote-ref-4)
5. READ COMMITTED: Nivel de Isolamento. Especifica que as instruções não podem ler dados que foram modificados, mas que ainda não foram confirmados por outras transações. Isso impede a ocorrencia de DirtyReads. Os dados podem ser alterados por outras transações entre instruções individuais dentro da transação atual, resultando em Unrepeatable Reads ou anomalias do tipo Phantom. É opção padrão do SQL Server *(in http://msdn.microsoft.com/)* [↑](#footnote-ref-5)
6. SERIALIZABLE: Especifica um nível de isolamento no qual as instruções não podem ler dados que foram modificados, e ainda não foram submetidos por outras transacções, que nenhuma outra transação pode modificar dados lidos pela transação atual até que a transação atual seja concluída e que aplica Lock ao Critério ddas operações de leitura. *(in http://msdn.microsoft.com/)* [↑](#footnote-ref-6)
7. LOCK ao Critério: Outras transações não podem inserir linhas novas com valores chave que estejam no intervalo de chaves lido por qualquer instrução da transação atual até que esta seja concluída. *(in http://msdn.microsoft.com/)* [↑](#footnote-ref-7)
8. PHANTOM READ: Manifesta-se quando uma transacção concorrente insere dados que cumprem um critério especificado pela transacção actual quando já houve uma leitura. Estes dados não estavam presentes no inicio da transacção mas poderão ser considerados mais tarde, introduzindo um potencial para a ocorrência de distorções na lógica transaccional prevista. *(in http://msdn.microsoft.com/)* [↑](#footnote-ref-8)
9. VIEW: Uma tabela Virtual cujo conteúdo é definido por uma Query. [↑](#footnote-ref-9)
10. FUNÇÃO: Uma Rotina em T-SQL que se caracteriza por resolver um cálculo ou uma query, recebendo um conjunto de parâmetros, e devolvendo um resultado, podendo este ser Escalar (um valor) ou uma Tabela. [↑](#footnote-ref-10)
11. Procedimento armazenado ou STORED PROCEDURE: Um procedimento que se caracteriza por receber um conjunto de parâmetros (alguns dos quais poderão ser alterados no decurso da execução do procedimento, desde que devidamente assinalados com a palavra OUT) e executar um conjunto de operações. [↑](#footnote-ref-11)