Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

Gestão de Ativos (Fase 1)

Autores: 46971 Diogo Fernandes
47612 Tiago Ribeiro

Relatório para a Unidade Curricular de Sistemas de Informação 1 da Licenciatura em Engenharia Informática e de Computadores

Professor: Afonso Remédios

04 – Dezembro – 2021

Resumo

Um Sistema de Gestão de Base de Dados (SGBD) visa a organização estruturada (modelo relacional) de dados de um derminado contexto (negócio), gerindo o seu armazenamento, manipulação e pesquisa dos dados, funcionando como um interface entre aplicações e os dados necessários para a sua execução.

Um SGBD procura evitar a redundância de Dados (informação armazenada em vários ficheiros), isolamento (ficheiros inacessíveis pelas aplicações) e inconsistência dos mesmos (cópias diferentes dos mesmos dados).

O modelo relacional é um modelo de dados que representa ou implementa adquadamente o SGBD. Baseia-se num princípio de que todos os dados estão armazenados em tabelas ou, no caso da matemática, de relações. Toda a sua definição é teórica, baseado na teoria de conjuntos e na lógica de predicados.

Abstract

A Relational Data Base Management System (RDBMS) aims at the structured data organization (relational model) for a particular context (business), managing its storage, manipulation and data querying, functioning as an interface between application data and its execution.

A RDBMS seeks to avoid data redundancy (information stored in several files), isolation (files inaccessible by applications) and inconsistency (different copies of the same data).

The relational model is a data model that adequately represents or implements the DBMS.

It is based on a principle that all data is stored in tables or, in the case of mathematics, of relations. Its entire definition is theoretical, based on set theory and predicate logic.

Índice

Lista de Figuras

Lista de Tabelas

1. Introdução

A Gestão de Ativos visa a obtenção, de forma integrada, de um balanço adequado entre o desempenho, o custo e o risco associados aos ativos, ao longo do seu ciclo de vida útil. A Gestão de Ativos pretende constituir-se como uma ferramenta de suporte à tomada de decisões de gestão, tendo por base informação de qualidade e o prévio estabelecimento dos níveis de serviço. A Maintain4ver é uma empresa de manutenção que pretende implementar um sistema de informação para a gestão de manutenção de ativos físicos. Pretende-se neste trabalho desenvolver o seu modelo de dados relacional e criar o seu respetivo modelo físico, com a integração de mecanismos de garantam a correta implementação do modelo e que permitam manipulação dos dados dentro de transações seguras de forma a evitar perda, ou leituras erradas, de informação.

2. Modelo de Dados

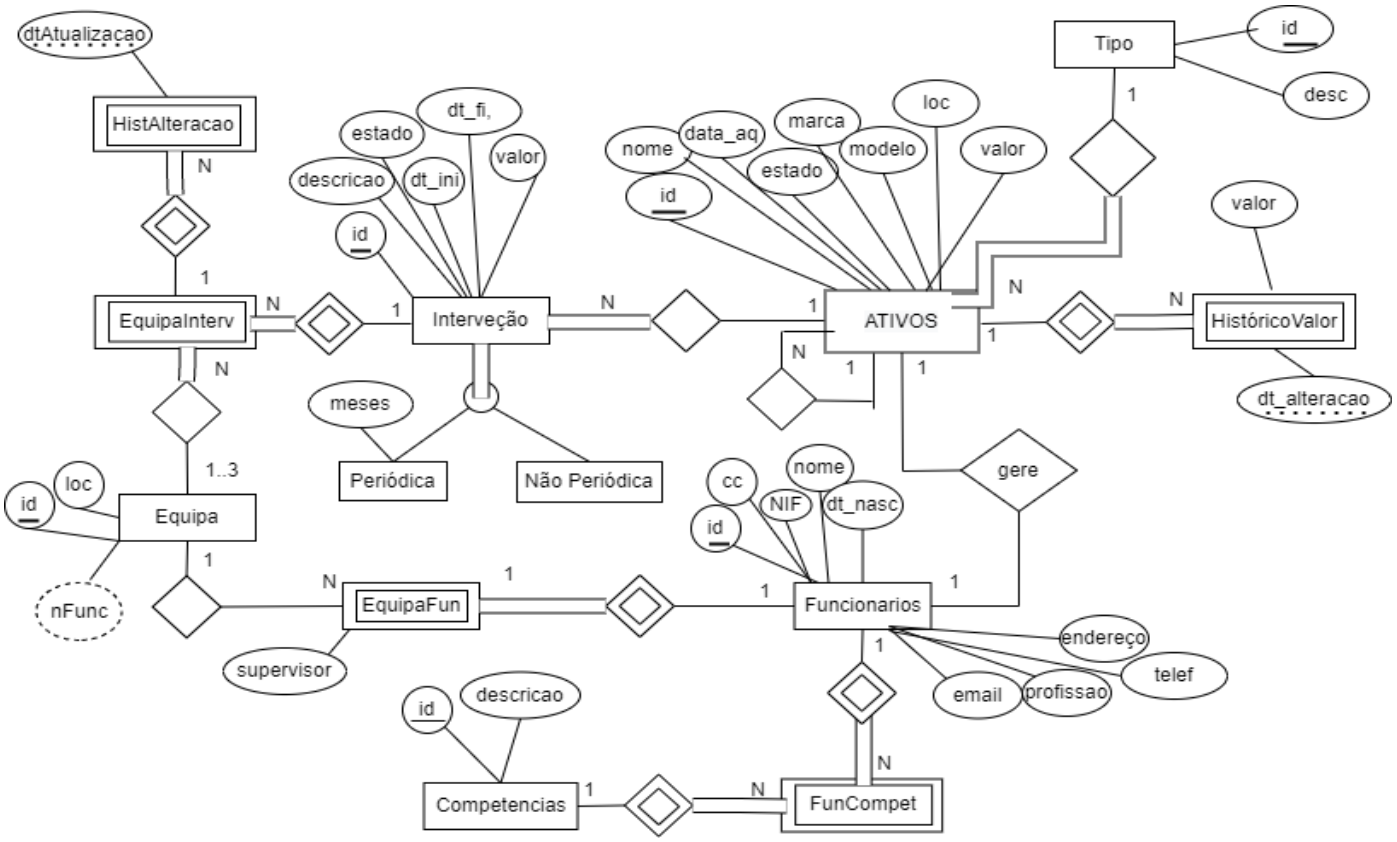

Figura 1- Diagrama ER

3. Detalhe do modelo de dados

O Modelo de Dados lógico do projeto proposto consiste em um conjunto de relações cujo detalhe se descreve nas proximas tabelas:

Tabela com registo das competências a serem atribuidas aos funcionários:

Atributo	Domínio	Restrição
<u>id</u>	int	PK
descricao	varchar	

Tabela 1 - Competências

Tabela com registo de todos os funcionários da empresa:

Atributo	Domínio	Restrição
id	int	PK
nome	varchar	
cc	int	UNIQUE e dimensão = 8
nif	int	UNIQUE e formato (“1_____” ou “2_____)
dtnasc	date	
endereco	varchar	
email	varchar	UNIQUE e formato '_@_._'
ntelefone	nvarchar	Formato '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'
profissao	varchar	

Tabela 2 – Funcionários

Foi criado um *trigger* que é lançado sempre que há inserção de um novo funcionario. Este *trigger*, adiciona à tabela **EquipaFunc** o novo funcionario, para mais tarde lhe ser atribuida um equipa.

```
CREATE OR ALTER  TRIGGER [dbo].[trg_InsertmembroEquipa]
ON [dbo].[Funcionarios]
FOR INSERT
AS
BEGIN
            DECLARE @funcId int
            SELECT @funcId = id FROM inserted
            INSERT INTO EquipaFunc VALUES (@funcId, null,null)
END
```

Figura 2 - Trigger da tabela Funcionários

Tabela de associação de competências aos funcionários:

Atributo	Domínio	Restrição
idFunc	int	PK / FK Ref. Funcionarios.id
idComp	int	PK / FK Ref. Competencias.id

Tabela 3 - FunCompet

Tabela para registar cada tipo associado a um ativo:

Atributo	Domínio	Restrição
id	int	PK
descricao	varchar	

Tabela 4 - Tipos

Tabela para registar cada ativo:

Atributo	Domínio	Restrição
id	int	PK
nome	varchar	
valor	money	
dtAquisicao	date	
estado	bit	
marca	varchar	
modelo	varchar	
localizacao	varchar	
parentId	int	FK Ref. Ativos.id
tipoId	int	FK Ref. Tipo.id
gestorId	int	FK Ref. Funcionarios.id

Tabela 5 - Ativos

Nesta tabela foi incorporado um *trigger* que é executado sempre que é inserido um novo ativo ou atualizado o seu valor. A sua função é inserir o valor do ativo e data de atualização na tabela **Historico**, de forma a manter o registo de todos os valores assumidos pelos ativos.

```
CREATE OR ALTER TRIGGER [dbo].[trg_HistoricoValorAtivo]
ON [dbo].[Ativos]
AFTER INSERT, UPDATE
AS
IF(update(valor))
BEGIN
            DECLARE @ativoValor money,
                    @ativoId int,
                @data datetime = GETDATE()
            SELECT @ativoId = id , @ativoValor = valor FROM inserted

    INSERT INTO dbo.Historico values (@ativoId,@data, @ativoValor)
END

go
-- insere na tabela de equipaInter de ativos
CREATE TRIGGER dbo.trg_insertInter
   ON  dbo.[Intervencao]
AFTER INSERT
AS
BEGIN
    declare @IntervencaoID int

    SELECT @IntervencaoID = id from inserted
    INSERT INTO EquipaIntervencao VALUES (@IntervencaoID,null)
END
```

Tabela 6 - Trigger de abastecimento da tabela Historico

Tabela de registo histórico do valor comercial do activo, em euros, registando-se a data (no formato dd-mm-aaaa) em que a alteração ocorreu.

Atributo	Domínio	Restrição
idAtivo	int	PK / FK Ref. Ativos.id
dtAlteracao	datetime	PK
valor	money	

Tabela 7 - Historico

Tabela de registo das equipas e quantidade de funcionários (nFunc) que constituem a equipa:

Atributo	Domínio	Restrição
id	int	PK
localizacao	varchar	
nFunc	int	

Tabela 8 - Equipa

Tabela de correspondência entre cada funcionário e equipa. Como cada funcionário só pode ter uma equipa, a relação entre esta tabela e Funcionários é de 1 - 1

Atributo	Domínio	Restrição
funcId	int	PK / FK Ref. Funcinarios.id
equipaId	int	FK Ref. Equipa.id
supervisor	int	FK Ref. Funcinarios.id

Tabela 9 - EquipaFunc

Foram adicionados dois *triggers*, que são responsáveis por manter o número de funcionarios de cada equipa (*nFunc* da tabela **Equipa**) actualizado, sempre que se executa uma actualização, inserção ou remoção de funcionarios. Quando é feita uma remoção, elimina também o registo do funcionário de todas as tabelas onde está referenciado

```
CREATE OR ALTER  TRIGGER [dbo].[trg_updateNfunc]
ON [dbo].[EquipaFunc]
after update
AS
BEGIN
            DECLARE @funcId int,
            @equipaID int,
            @supervisorID int,
            @nFunc INT ,
            @nFuncSup INT

            SELECT @funcId = funcId , @equipaID = equipaID , @nFunc = ef.CONT , @nFuncSup = CONTS , @supervisorID = supervisor
            FROM inserted
                LEFT JOIN (SELECT EquipaId as eid,COUNT(1) AS CONT from EquipaFunc GROUP BY EquipaId )ef on ef.eid = equipaId
                LEFT JOIN (SELECT EquipaId as eids,COUNT(DISTINCT supervisor) AS CONTS from EquipaFunc GROUP BY EquipaId )
                                                                                efs on efs.eids = equipaId
            Update Equipa set nFunc = ISNULL(@nFunc,0) + iSNULL(@nFuncSup,0) where id = @equipaID
            UPDATE EquipaFunc set equipaId = @equipaID, supervisor = @supervisorID where funcId = @supervisorID

            SELECT @funcId = funcId , @equipaID = equipaID , @nFunc = ef.CONT , @nFuncSup = CONTS
            FROM deleted
                LEFT JOIN (SELECT EquipaId as eid,COUNT(1) AS CONT from EquipaFunc GROUP BY EquipaId )ef on ef.eid = equipaId
                LEFT JOIN (SELECT EquipaId as eids,COUNT(DISTINCT supervisor) AS CONTS from EquipaFunc GROUP BY EquipaId )
                                                                                efs on efs.eids = equipaId
            Update Equipa set nFunc = ISNULL(@nFunc,0) + iSNULL(@nFuncSup,0) where id = @equipaID
END
```

Figura 3 – Trigger ativo a cada update na tabela EquipaFunc

```
CREATE OR ALTER TRIGGER [trg_deletefunc]
ON [dbo].[EquipaFunc]
FOR DELETE
AS
BEGIN
    declare @funcionario int,
            @equipa int,
            @supervidorId int,
            @nFunc INT ,
            @nFuncSup INT

    SELECT @funcionario = funcId , @equipa = equipaID , @nFunc = ef.CONT , @nFuncSup = CONTS
    FROM deleted
                LEFT JOIN (SELECT EquipaId as eid,COUNT(1) AS CONT from EquipaFunc GROUP BY EquipaId )ef on ef.eid = equipaId
                LEFT JOIN (SELECT EquipaId as eids,COUNT(DISTINCT supervisor) AS CONTS from EquipaFunc GROUP BY EquipaId )
                                                                                efs on efs.eids = equipaId
    Update Equipa set nFunc = ISNULL(@nFunc,0) + iSNULL(@nFuncSup,0) where id = @equipa


    UPDATE EquipaFunc SET @supervidorId = NULL WHERE supervisor = @funcionario
    DELETE FROM EquipaFunc WHERE funcId = @funcionario
    DELETE FROM Funcionarios WHERE id = @funcionario
    DELETE FROM FunCompet WHERE idFunc = @funcionario


END
```

Figura 4 - Trigger ativo a cada delete na tabela EquipaFunc

Tabela para registar todas as intervenções:

Atributo	Domínio	Restrição
<u>id</u>	int	PK
descricao	varchar	Aceita valores “avaria”, “rutura” ou “inspecao”
estado	varchar	Aceita valores “por atribuir”, “em análise”, “em execução” ou “concluido”
dtInicio	date	
dtFim	date	Data superior a dtInicio
valor	money	
ativoId	int	FK Ref. Ativos.id

Tabela 10 - Intervencao

O *trigger* `trg_updateEstado`, presente na tabela intervencao, é executado no momento da tentativa de actualização do estado da intervenção. Este *trigger* garante que uma equipa não tem mais do que uma intervenção atribuida no estado “em execução”. Poderiamos também ter criado um *trigger* para validar a cada inserção de dados se a data de inicio era superior à data do ativo, mas como existe um procedure para inserir intervenções optamos por fazer essa validação neste.

Criámos também o *trigger* `trg_insertInter`, que insere na tabela EquipaIntervencao um registo com o id da intervenção e uma equipa a null (por atribuir).

```
CREATE OR ALTER TRIGGER [dbo].[trg_updateEstado]
ON [dbo].[Intervencao]
AFTER UPDATE
AS
BEGIN
            DECLARE @Intervencao int,
            @Equipa int
            SELECT @Intervencao = id, @Equipa = ei.equipaId from inserted
                left join EquipaIntervencao ei on ei.idIntervencao = id
IF Update(estado)
    BEGIN
        if ((select count(I.ID)
            from EquipaIntervencao ei
            inner join Intervencao i ON i.id = ei.idIntervencao and i.estado = 'em execução'
            where equipaId = @Equipa) > 1)
            BEGIN
                RAISERROR ('Equipa já tem uma intervenção em execução', 16, 1);
            END
    END
END
```

Figura 5 - trigger trg_updateEstado

```
CREATE TRIGGER dbo.trg_insertInter
   ON  dbo.[Intervencao]
AFTER INSERT
AS
BEGIN
    declare @IntervencaoID int

    SELECT @IntervencaoID = id from inserted
    INSERT INTO EquipaIntervencao VALUES (@IntervencaoID,null)
END
```

Figura 6 - Trigger insertInter

Tabela com registo das intervenções periódicas e registo dos meses:

Atributo	Domínio	Restrição
id	int	PK / FK Ref. Intervencao.id
meses	varchar	

Tabela 11 - IntervencaoPeriodica

Tabela de registo para associar uma equipa a cada Intervenção:

Atributo	Domínio	Restrição
idIntervencao	int	PK / FK Ref. Intervencao.id
equipaId	int	FK Ref. Equipa.id

Tabela 12 - EquipaIntervencao

```
CREATE OR ALTER TRIGGER [dbo].[trg_UpdateEquipaIntervencao]
ON [dbo].[EquipaIntervencao]
AFTER UPDATE
AS
BEGIN
            DECLARE @Intervencao int,
            @Equipa int,
            @Data datetime = GETDATE()
            SELECT @Intervencao = idIntervencao, @Equipa = equipaId FROM inserted
--não pode ter menos de dois funcionarios sendo um deles supervisor
IF EXISTS (
            Select *
            from [EquipaIntervencao] ei
                inner join inserted i on i.idIntervencao = ei.idIntervencao
                LEFT JOIN Equipa e on e.id = ei.equipaId
            where e.nFunc < 2 OR E.id IN (SELECT equipaId from EquipaFunc where supervisor = 0)
            )
    BEGIN
        RAISERROR ('Equipa tem de ter pelo menos 2 funcionarios sendo um deles o supervisor', 16, 1);
    END

--não pode ter mais de 3 intervenções atribuidas e apenas uma em curso

IF ( (select COUNT (ei.equipaId)
     from EquipaIntervencao ei
     inner join inserted on ei.equipaId = inserted.equipaId
     inner join Intervencao i ON I.id = ei.idIntervencao and i.estado in ('em análise','em execução')) > 3)
    BEGIN
        RAISERROR ('Limite de intervençoes atingida', 16, 1);
    END
```

```
IF ( (select COUNT (ei.equipaId)
     from EquipaIntervencao ei
     inner join inserted on ei.equipaId = inserted.equipaId
     inner join Intervencao i ON I.id = ei.idIntervencao and i.estado in ('em execução')) >= 2)

    BEGIN
        RAISERROR ('Limite de intervençoes em execucao atingida', 16, 1);
    END


--a pessoa que gere o ativo nao pode participar na intervencao
if exists (
             SELECT DISTINCT funcId from EquipaFunc
            inner join Ativos on Ativos.gestorId = EquipaFunc.funcId
            inner join Intervencao on Intervencao.ativoId = Ativos.id AND Intervencao.id = @Intervencao
            inner join inserted on inserted.equipaId = EquipaFunc.equipaId
            )
    BEGIN
        RAISERROR ('Pessoa que gere o ativo nao pode pertencer à equipa', 16, 1);
    END

--a descricao da intervencao tem de ser compativel com  as competencias

IF NOT EXISTS (
           select descricao
           from equipaFunc EF
           inner join FunCompet FC on FC.idFunc = EF.funcId
           inner join Competencias c on c.id = FC.idComp
           INNER JOIN inserted I ON I.equipaId = EF.equipaId
           where EF.equipaId = @Equipa and descricao in (SELECT descricao FROM Intervencao where id = @Intervencao)
           group by EF.equipaId, descricao
)
    BEGIN
        RAISERROR ('Equipa nao tem competencias', 16, 1);
    END

-- EQUIPA ATRIBUIDA
           update Intervencao set estado = 'em análise' where Intervencao.id = @Intervencao
           insert into HistAlteracaoEqInterv VALUES (@Intervencao, @Equipa, @Data)
END
```

Figura 7 - Trigger UpdateEquipaIntervencao

Criamos o *trigger* `[trg_UpdateEquipaIntervencao]`, que a cada update de equipa, valida as restrições dadas no enunciado. Caso estas sejam respeitadas, é feita uma inserção de registo na tabela HistAlteracaoEqInterv com a data da alteração.

Também criámos um *trigger* para quando for feito o delete de uma intervenção, elimina os registos da tabela de histórico e das intervenções.

```
--Elimina da tabela equipa Intervençao e
CREATE OR ALTER TRIGGER [dbo].[trg_DeleteEquipaIntervencao]
ON [dbo].[EquipaIntervencao]
FOR DELETE
AS
BEGIN
            DECLARE @Intervencao int,
            @Equipa int,
            @Data datetime = GETDATE()
            SELECT @Intervencao = idIntervencao , @Equipa = equipaId FROM deleted

        DELETE FROM dbo.HistAlteracaoEqInterv where @Intervencao = idIntervencao
        DELETE FROM dbo.IntervencaoPeriodica where @Intervencao = id
        DELETE FROM dbo.Intervencao where @Intervencao = id
END
```

Figura 8 – Trigger `DeleteEquipaIntervencao`

Tabela de registo de histórico de todas as equipas intervenientes numa intervenção. Esta tabela é abastecida automaticamente por intermédio do *trigger* UpdateEquipaIntervencao da tabela EquipaIntervenção.

Tabela 13 - HistAlteracaoEqInterv

Atributo	Domínio	Restrição
idIntervencao	int	PK/ FK Ref. Intervencao.id
dtAtualizacao	datetime	PK
equipaId	int	

4. Resolução dos problemas propostos

Alinea 2d- Mecanismo que permite inserir, remover e atualizar informação de uma pessoa ser usar *insert*, *update* e *delete* directos.

Para a resolução desta alínea optamos pela implementação de um *Stored-procedure* visto que é o mais adequado para afetação de dados na BD. Este mecanismo recebo os dados do funcionário e o tipo de operação e a realizar (*insert*, *update* ou *delete*).

```
CREATE OR ALTER PROCEDURE [dbo].[SP_Funcionarios] (@id int,
                                        @nome VARCHAR(50),
                                        @cc int,
                                        @nif int,
                                        @dtNasc date,
                                        @endereco VARCHAR(50),
                                        @email VARCHAR(50),
                                        @ntelefone VARCHAR(50),
                                        @profissao VARCHAR(50),
                                        @operationType NVARCHAR(15))
AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRANSACTION
BEGIN
    IF @operationType = 'Insert'
    BEGIN
        INSERT INTO Funcionarios(id, nome, cc, nif, dtNasc, endereco, email, ntelefone, profissao)
        VALUES (@id, @nome, @cc, @nif,@dtNasc, @endereco, @email, @ntelefone, @profissao)
    END
    IF @operationType = 'Update'
    BEGIN
        UPDATE Funcionarios
            SET    nome = @nome,
                   cc = @cc,
                   nif = @nif,
                   dtNasc = @dtNasc,
                   endereco = @endereco,
                   email = @email,
                   ntelefone = @ntelefone,
                   profissao= @profissao
            WHERE  id = @id
    END
    ELSE IF @operationType = 'Delete'
    BEGIN
        DELETE FROM EquipaFunc WHERE funcId = @id
    END
COMMIT
END
```

Figura 9- Procedure SP_Funcionarios

Alinea 2e- Obter o código de uma equipa livre, dada uma descrição de intervenção, capaz de resolver o problema. Em caso de haver várias equipas deve escolher-se a que teve uma intervenção atribuída há mais tempo.

Neste caso foi implementada uma função que recebe uma determinada competencia e devolve o id da equipa disponivel, tendo em conta vários fatores, como por exemplo, não ter uma intervenção atribuida com o estado “em execução”. A validação da competencia é realizada consante as competencias do funcionario que a compõem.

```
CREATE or alter FUNCTION dbo.F_ObterEquipaLivre (@competencia varchar(50))
RETURNS INT
AS
BEGIN
    declare @equipa int

        ;with cte as (
        SELECT TOP 1 e.equipaId AS EQ, COUNT(e.idIntervencao) as nIntervencoes, maxData
        FROM EquipaIntervencao E
            INNER JOIN Intervencao I ON I.id = E.idIntervencao
            inner join (SELECT equipaId as eq, max(dtAtualizaco) as maxData
                        FROM HistAlteracaoEqInterv
                        group by equipaId) h on h.eq = E.equipaId
        where e.equipaId is not NULL AND I.estado != 'em execução'
                and E.equipaId IN ( SELECT distinct equipaId
                                    FROM dbo.EquipaFunc e
                                        JOIN dbo.FunCompet f ON e.funcId = f.idFunc
                                        JOIN dbo.Competencias c ON c.id = f.idComp
                                        WHERE c.descricao like @competencia and e.equipaId is not null)
        group by e.equipaId,maxData
        HAVING COUNT(e.idIntervencao) < 3
        ORDER BY maxData ASC

        )
        SELECT @equipa = cte.EQ FROM cte
RETURN @equipa
end
```

Figura 10- Função ObterEquipaLivre

Alinea 2f- Procedimento que permite criar uma intervenção.

Este procedimento recebe os dados da nova intervenção e insere-a na tabela **Intervenção** e na tabela **EquipaIntervencao**, ficando inicialmente sem equipa atribuida.

```
create or ALTER  PROCEDURE [dbo].[SP_criaInter]
    @id int,
    @descricao varchar(50),
    @dtInicio date,
    @dtFim date,
    @valor money,
    @ativoId int,
    @meses int

AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED  --Apenas estou a ler uma vez
BEGIN TRANSACTION
if ((SELECT dtAquisicao FROM DBO.Ativos WHERE ativos.id = @ativoId) < @dtInicio)
    begin

        INSERT INTO dbo.Intervencao VALUES(@id,@descricao,'por atribuir',@dtInicio,@dtFim,@valor,@ativoId)
        INSERT INTO dbo.EquipaIntervencao VALUES (@id,null)
        if (@meses > 0)
            INSERT INTO dbo.IntervencaoPeriodica values (@id, @meses)
        COMMIT
    END
ELSE
    BEGIN
        RAISERROR('Data da Intervencao menor que a data de aquisicao, linha não foi inserida',16,1)
        ROLLBACK
    END
```

Figura 11- Procedure SP_CriarInter

Alinea 2g- Mecanismo que permite criar uma equipa.

O mecanismo implementado foi um *Stored-procedure*, que recebe como parametros os dados da equipa e insere uma nova equipa na tabela **Equipa**.

```
CREATE OR ALTER PROCEDURE [dbo].[SP_criaEquipa] (@idEquipa int, @localizacao varchar(50))
AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN
    BEGIN TRANSACTION
        INSERT INTO dbo.Equipa VALUES (@idEquipa, @localizacao, 0)
    COMMIT
END
```

Figura 12- Procedure SP_CriarEquipa

Alinea 2h- Actualizar os elementos de uma equipa e associar as respetivas competencias.

Esta solução é realizada através de um *Stored-procedure* que recebe o id da equipa, os dados do funcionario e a operação a efetuar. São feitas algumas verificações antes da afetação, tais como se o funcionario ou equipa existe e a associação das competencias à equipa é obtida pelas competencias dos funcionarios que a compõem.

```
CREATE OR ALTER PROCEDURE [dbo].[SP_ActualizarElementosEquipa] (@equipaId int, @FuncId int, @operationType varchar(20), @supervisor INT)
AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN

IF (not EXISTS (SELECT * FROM EquipaFunc WHERE funcId = @FuncId) or NOT EXISTS (SELECT * FROM Equipa WHERE id = @equipaId))
    BEGIN
        RAISERROR('Funcionario/Equipa nao existe',16,1)
    END
ELSE
    BEGIN
        BEGIN TRANSACTION
        IF (@operationType = 'Insert')
            BEGIN
                IF (@supervisor = 0)
                    UPDATE dbo.EquipaFunc SET equipaId = @equipaId WHERE funcId = @FuncId
                else
                    begin
                        UPDATE dbo.EquipaFunc SET equipaId = @equipaId, supervisor = @supervisor WHERE funcId = @FuncId
                    end
            END
        IF (@operationType = 'Delete')
            BEGIN
                IF (@supervisor = 0)
                        UPDATE dbo.EquipaFunc SET equipaId = NULL, supervisor = NULL WHERE funcId = @FuncId
                else
                    begin
                        UPDATE dbo.EquipaFunc SET equipaId = nuLL, supervisor = NULL WHERE funcId = @FuncId
                        UPDATE dbo.EquipaFunc SET supervisor = NULL WHERE supervisor = @supervisor
                    end
            END
        COMMIT
    END
```

Figura 13- Procedure SP_ActualizarElementosEquipa

Alinea 2i- Criar uma função para produzir a listagem (código, descrição) das intervenções de um determinado ano.

Esta função retorna uma tabela com todas as intervenções num determinado ano, sendo este recebido como parametro.

```
CREATE OR ALTER FUNCTION dbo.IntervencaoAno (@year int)
RETURNS TABLE
AS
RETURN (SELECT i.id, i.descricao FROM dbo.Intervencao i
        WHERE YEAR(i.dtInicio) = @year AND YEAR(i.dtFim) = @year)
```

Figura 14- Função IntervencaoAno

Alinea 2j- Actualizar estado de uma intervenção.

Esta operação é efetuada através de um *Stored-procedure* que recebe o id da intervençao e o seu novo estado, e se essa intervenção existir é feita a respetiva afetação.

```
CREATE OR ALTER PROCEDURE DBO.SP_AtualizarEstadoIntervencao (@intervencaoID int, @novoEstado varchar(20) )
AS
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN
    if exists (select id froM Intervencao where id = @intervencaoID)
        begin
            BEGIN TRANSACTION
            BEGIN TRY
                UPDATE Intervencao SET estado = @novoEstado WHERE id = @intervencaoID
            end try
            begin catch
                rollback transaction
            end catch
            COMMIT
        end
    ELSE
        RAISERROR ('Intervencao nao existe', 16, 1);
END
```

Figura 15- Procedure SP_ActualizarEstadoIntervenção

Alinea 2j- Criar uma vista que mostre o resumo das intervenções e que possibilita a alteração do estado de uma ou mais intervenções.

A criação da vista consiste em apresentar todas as intervenções e seus ativos. Para a alteração do estado de varias intervenções, optamos por incorporar um *trigger*, que itera sobre todos os registos do estado da intervenção, atraves de um cursor, e altera-os com o novo valor.

```
CREATE OR  ALTER TRIGGER [dbo].[trg_viewResumoInterv]
ON dbo.TV_ResumoIntervencoes
instead of update
as
BEGIN
    IF UPDATE (estadoIntervencao)
        BEGIN
            DECLARE @interven int,
                    @estado varchar(20)
            declare c cursor for
                select IdIntervencao , estadoIntervencao FROM inserted
            open c
            fetch next from c into @interven, @estado
            while @@FETCH_STATUS = 0
            BEGIN
                        begin try
                            UPDATE Intervencao SET estado = @estado Where id = @interven
                        end try
                        begin catch
                            DECLARE @Exception varchar (5000)
                            select @Exception = ERROR_MESSAGE()
                            PRINT (@Exception)
                        end catch
            fetch next from c into @interven, @estado
            end
            close c
            deallocate c
        END
    else
        RAISERROR ('Só é possivel alterar o estado da Intervencao', 16, 1);
END
```

Figura 16- Vista ResumoInterv

5. Scripts com resolução dos problemas propostos

Em anexo a este relatório foram criados 5 Scripts:

1. CreateTables.sql -> Criar o modelo físico e triggers associados às tabelas;
2. DropTables.sql -> Remover o modelo físico;
3. InsertsTestValues.sql -> Preenche a base de dados com registos de teste;
4. Script2d_2k.sql -> com código da resolução das alineas 2d a 2k
5. Testes.sql -> Script com os testes realizados e respetivas descrições de testes.

6. Conclusão

Neste trabalho foram criadas relações com base nos fundamentos do modelo relacional. Foram usados os mecanismos de triggers para despoltar uma acção automática de acordo com o evento de um tabela e assim facilitar as restrinções do negócio. Foram usadas também *Store Procedures* de modo a facilitar os processos de transação de dados para as tabelas.

7. Referências

[1] Enunciado da primeira fase do Trabalho Prático da disciplica de Sistemas de Informação 2