universidade de aveiro deti departamento de electrónica, telecomunicações e informática

Projeto Base de Dados

Licenciatura em Engenharia Informática

2020/2021 2º Semestre

Alexandre Pinto 98401

Pedro Jorge 98418

Conteúdos

Introdução 3

Análise de Requisitos 4

Diagrama Entidade - Relação 5

Diagrama Relacional 6

SQL DDL 7

SQL Stored Procedures 8

SQL UDF 13

Normalização 14

Conclusão 15

Bibliografia 16

Introdução

No projeto final da Unidade Curricular de Base de Dados, com vista a reunir todas as competências lecionadas nas aulas práticas e teórico-práticas da disciplina, foi pedido aos alunos que escolhessem um tema (de carácter livre).

Desta forma, o tema escolhido e apresentado neste relatório passa pela gestão de uma clínica veterinária. A plataforma denomina-se de “HappyPetVet” e vai permitir tratar de entidades como: veterinário, enfermeiro, episódio (que inclui exame, consulta e cirurgia), receita, fatura, medicamentos, internamento e documento do animal.

Para a criação da base de dados foi usado o software SQL Server, enquanto que para a construção da interface gráfica foi usado o Visual Basic.Net.

O trabalho foi realizado em grupos de dois alunos, com data de entrega prevista para 24 de junho de 2021.

Análise de Requisitos

De maneira a criar um programa que permita gerir uma clínica veterinária, pretende-se implementar uma plataforma onde todos os utensílios de gestão estejam facilmente acessíveis.

Para tal, a clínica veterinária irá conter um número indefinido de veterinários, enfermeiros, animais, episódios (consultas, exames, cirurgias e internamentos) e clientes:

- Os veterinários podem ser inseridos, editados e eliminados e irão realizar consultas, exames e podem prescrever receitas e internamentos;

- Os enfermeiros podem ser inseridos, editados e eliminados e irão auxiliar o veterinário a realizar consultas e exames;

- Os animais podem ser inseridos, editados e eliminados. Podem ser prescritos consultas, exames, receitas e internamentos.

- Os episódios podem ser inseridos, editados e eliminados. Cada episódio pode ser uma consulta, cirurgia, exame ou internamento.

- Os clientes podem ser inseridos, editados e eliminados

Diagramas Entidade – Relação

Apesar de os diagramas já terem sido entregues ao docente da cadeira, numa primeira entrega, estes sofreram algumas alterações aquando da construção da base de dados em questão:

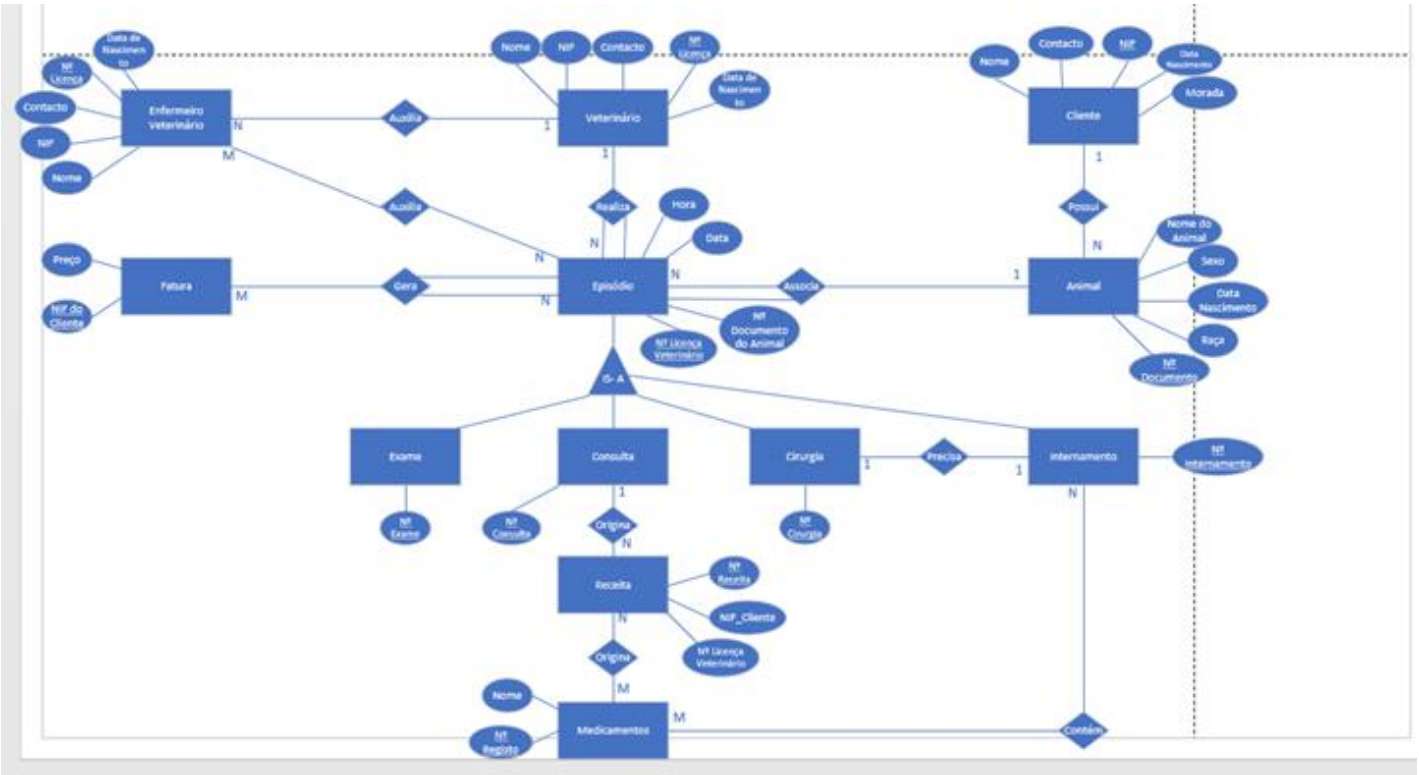
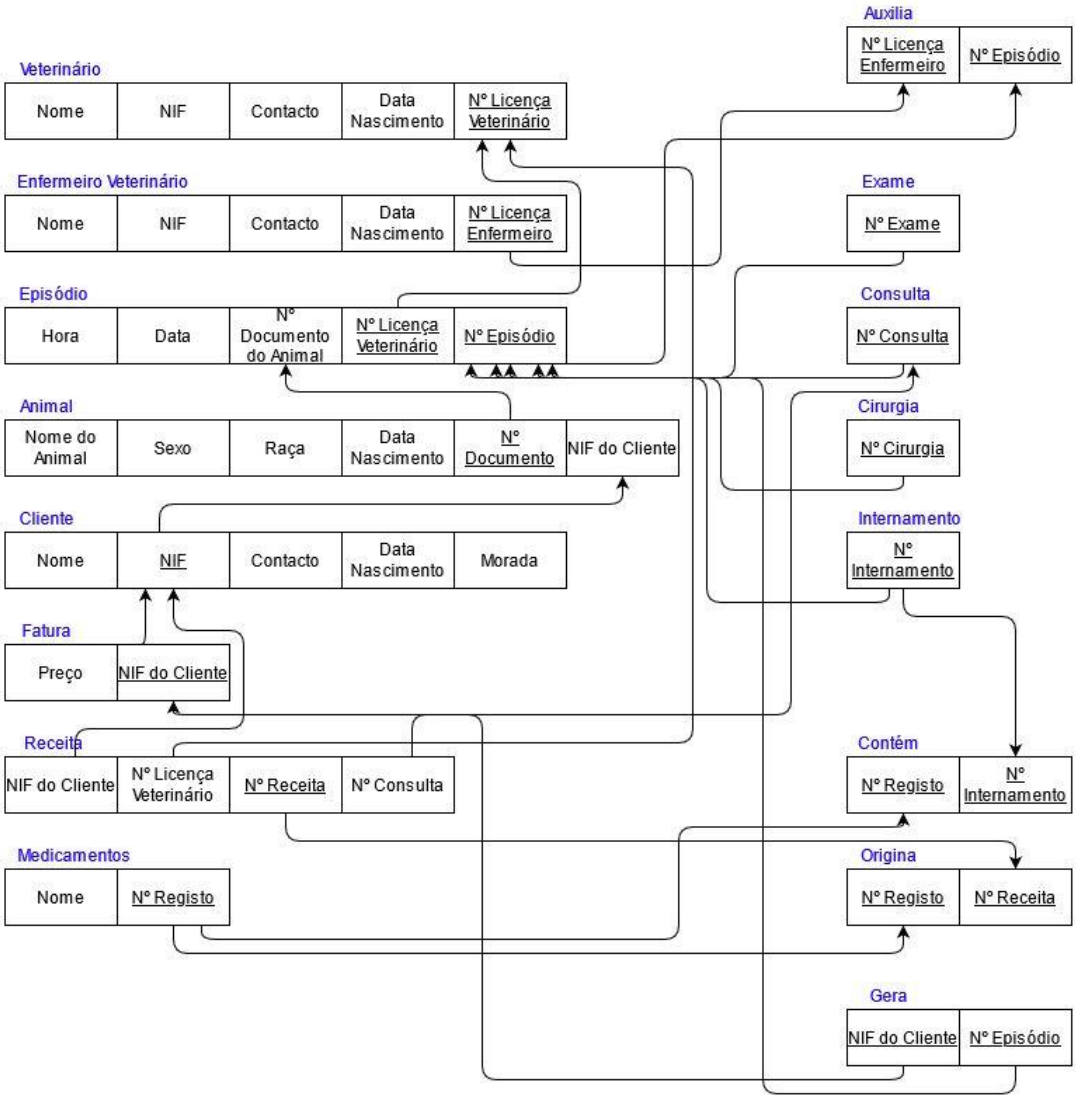


Diagrama Relacional



SQL DDL

Inicialmente, as tabelas foram criadas tendo em conta o diagrama inicial. À medida que foram surgindo mais necessidades, foi preciso alterar algumas *tables*.

As *tables* HEALTHYPETVET.VETERINARIO, HEALTHYPETVET.ENFERMEIRO, HEALTHYPETVET.ANIMAL, HEALTHYPETVET.MEDICAMENTOS, HEALTHYPETVET.EPISODIO e HEALTHYPETVET.RECEITA contêm uma coluna do tipo IDENTITY que permite auto-gerar um número de serviço para cada uma, que será a *primary key* de cada uma das *tables*.

No caso das *foreign key*, utilizamos os comandos ON UPDATE CASCADE e ON DELETE CASCADE. Assim, se uma entidade maior for eliminada ou editada, os dados que estão nas outras tabelas também serão atualizados ou eliminados.

Os dados relativamente às DDL estão no ficheiro Tables.sql.

Os dados inseridos nas tabelas estão no ficheiro Values.sql.

SQL STORED PROCEDURES

Ao longo do nosso projeto, fomos utilizando bastantes *stored procedures*. Estes procedimentos foram utilizadas para diferentes fins de modo a que fosse possível filtrar os dados e eliminar possíveis erros ao inserir, atualizar e eliminar dados das tabelas da base de dados:

- inserir dados - INSERT
- editar dados - UPDATE
- eliminar dados - DELETE

Relativamente a inserir dados, o procedimento de inserção utilizado nas entidades Veterinário, Enfermeiro, Cliente e Animal é bastante semelhante. O *Stored Procedure* tenta inserir dados nas tabelas da base de dados utilizando o comando INSERT, mas, caso isso não seja possível e ocorra algum erro, a inserção não vai ocorrer. Para isto, utilizamos um TRY…CATCH.

```
--DROP PROCEDURE HEALTHYPETVET.p_insertVet
CREATE PROCEDURE HEALTHYPETVET.p_insertVet (
        @nome varchar(150),
        @nif int,
        @contacto int,
        @dataNasc DATE)
AS
BEGIN TRAN
    BEGIN TRY
        INSERT INTO HEALTHYPETVET.VETERINARIO VALUES(@nome, @nif, @contacto, @dataNasc)
    END TRY
    BEGIN CATCH

    END CATCH
COMMIT TRAN;
```

Para a entidade Episódio, o seu procedimento vai funcionar de uma forma diferente. Uma Consulta, Cirurgia, Exame ou Internamento são Episódios, logo há uma relação entre estas

entidades. Quando adicionamos um Episódio temos de escolher qual é o tipo de Episódio entre as 4 opções referidas. Por isso, o procedimento de inserção de dados relativos aos episódios vai tentar adicionar os dados à tabela de Episódios e, caso isso seja possível, vai buscar o número do episódio gerado automaticamente e adicionar à tabela da entidade escolhida anteriormente. Caso não seja possível inserir dados na primeira tabela, já não será adicionado nenhum dado na segunda. Foram utilizados alguns comandos IF para filtrar as opções e TRY...CATCH para evitar erros possíveis.

```
--DROP PROCEDURE HEALTHYPETVET.p_insertEpi
CREATE PROCEDURE HEALTHYPETVET.p_insertEpi (
        @hora time,
        @dataEpi DATE,
        @NDocAni int,
        @NLicencaVet int,
        @opcao varchar(150))
AS
BEGIN TRAN
    BEGIN TRY
        INSERT INTO HEALTHYPETVET.EPISODIO VALUES(@hora, @dataEpi, @NDocAni, @NLicencaVet)
        DECLARE @NEpisodio AS int
        SET @NEpisodio = (SELECT TOP 1 NEpisodio FROM HEALTHYPETVET.EPISODIO ORDER BY NEpisodio DESC)
        IF  (@opcao = 'Consulta')
            INSERT INTO HEALTHYPETVET.CONSULTA_EPISODIO VALUES(@NEpisodio)
        IF  (@opcao = 'Cirurgia')
            INSERT INTO HEALTHYPETVET.CIRURGIA_EPISODIO VALUES(@NEpisodio)
        IF  (@opcao = 'Internamento')
            INSERT INTO HEALTHYPETVET.INTERNAMENTO_EPISODIO VALUES(@NEpisodio)
        IF  (@opcao = 'Exame')
            INSERT INTO HEALTHYPETVET.EXAME_EPISODIO VALUES(@NEpisodio)
    END TRY
    BEGIN CATCH

    END CATCH
COMMIT TRAN;
```

Relativamente a editar dados, em todas as entidades há dados onde não é possível alterar o seu valor. No caso dos Veterinários, Enfermeiros e Clientes não será possível alterar o NIF e a Data de Nascimento uma vez que estes dois valores não se alteram com o passar dos anos. Os restantes dados destas entidades são atualizados utilizando o comando UPDATE onde vão ser

atualizados os dados correspondentes à linha da tabela que contém o NIF introduzido no procedimento. Quando atualizados os dados, a maioria das tabelas onde a FOREIGN KEY contém alguns destes dados são atualizadas, umas vez que foi utilizado o comando ON UPDATE CASCADE.

```
--DROP PROCEDURE HEALTHYPETVET.p_editVet
CREATE PROCEDURE HEALTHYPETVET.p_editVet (
        @nome varchar(150),
        @nif int,
        @contacto int,
        @dataNasc DATE)
AS
BEGIN TRAN
    BEGIN TRY
        UPDATE HEALTHYPETVET.VETERINARIO SET Nome = @nome, Contacto = @contacto  WHERE NIF = @nif
    END TRY
    BEGIN CATCH

    END CATCH
COMMIT TRAN;
```

Quanto aos animais, não será possível alterar a sua Data de Nascimento nem o Número do Documento do Animal e a sua eliminação é feita através desse número no seu documento.

```
--DROP PROCEDURE HEALTHYPETVET.p_editAni
CREATE PROCEDURE HEALTHYPETVET.p_editAni (
                @nome varchar(150),
                @sexo char(1),
                @raca varchar(150),
                @nDoc int,
                @nif int)

AS
BEGIN TRAN
        BEGIN TRY
                UPDATE HEALTHYPETVET.ANIMAL SET Nome = @nome, Sexo = @sexo, Raca = @raca, NIFCliente = @nif WHERE NDocumento = @nDoc
                END TRY
        BEGIN CATCH

        END CATCH
COMMIT TRAN;
```

Quanto aos animais, não será possível alterar o Número do Episódio e a sua eliminação é feita através desse mesmo número.

```
--DROP PROCEDURE HEALTHYPETVET.p_editEpi
CREATE PROCEDURE HEALTHYPETVET.p_editEpi (
        @hora time,
        @dataEpi DATE,
        @NDocAni int,
        @NLicencaVet int,
        @nEpisodio int)
AS
BEGIN TRAN
    BEGIN TRY
        UPDATE HEALTHYPETVET.EPISODIO SET Hora = @hora, DataEpi = @dataEpi, NDocumentoAnimal = @NDocAni, NLicencaVet = @NLicencaVet WHERE NEpisodio = @nEpisodio
    END TRY
    BEGIN CATCH

    END CATCH
COMMIT TRAN;
```

Relativamente a eliminar dados, os procedimentos criados são muito simples. No caso das entidades Veterinário, Enfermeiro e Cliente, vamos introduzir o NIF da pessoa que queremos eliminar e o procedimento vai eliminar a linha da tabela cujo NIF é igual ao que foi dado nessa mesma entidade. Foi utilizado o comando DELETE. Todas as FOREIGN KEYs criadas nas tabelas das entidades e relações que contenham os dados da entidade que foi eliminada, vão ser eliminadas automaticamente, uma vez que a maioria destas chaves foram criadas como ON DELETE CASCADE.

```
--DROP PROCEDURE HEALTHYPETVET.p_deleteVet
CREATE PROCEDURE HEALTHYPETVET.p_deleteVet (
        @nif int)
AS
BEGIN TRAN

    BEGIN TRY
        DELETE FROM HEALTHYPETVET.VETERINARIO WHERE NIF = (
            SELECT NIF FROM HEALTHYPETVET.VETERINARIO AS Table1 WHERE Table1.NIF = @nif
            );
    END TRY
    BEGIN CATCH

    END CATCH

COMMIT TRAN;
```

Quanto às entidades Animal e Episódios, o funcionamento dos procedimentos é semelhante. No entanto, no caso dos animais, temos de introduzir o número do documento do animal. No caso do episódio, temos de introduzir o número do episódio que queremos eliminar.

Os dados relativamente às SPs estão no ficheiro Procedures.sql.

SQL USER DEFINED FUNCTIONS

No que toca às UDF´s, estas foram usadas para se poder filtrar o sexo e raça dos animais. Podemos escolher a raça e sexo que queremos utilizar na nossa filtragem e o procedimento vai buscar à tabela dos animais todos os animais cuja raça e o sexo são os iguais aos pretendidos.

```
--DROP FUNCTION getAnimaisPorRaca
CREATE FUNCTION getAnimaisPorRaca(@raca VARCHAR(150)) RETURNS Table AS
    RETURN(SELECT *
           FROM HEALTHYPETVET.ANIMAL AS Table1
           WHERE Table1.Raca = @raca)
```

UDF usada para filtrar o animal por raça

```
--DROP FUNCTION getAnimaisPorSexo
CREATE FUNCTION getAnimaisPorSexo(@sexo CHAR(1)) RETURNS Table AS
    RETURN(SELECT *
           FROM HEALTHYPETVET.ANIMAL AS Table1
           WHERE Table1.Sexo = @sexo)
```

UDF usada para filtrar o animal por sexo.

```
--DROP FUNCTION getAnimaisPorSexoRaca
CREATE FUNCTION getAnimaisPorSexoRaca(@raca VARCHAR(150), @sexo CHAR(1)) RETURNS Table AS
    RETURN(SELECT *
           FROM HEALTHYPETVET.ANIMAL AS Table1
           WHERE Table1.Raca = @raca AND Table1.Sexo = @sexo)
```

UDF usada para filtrar o animal por sexo e por raça.

Implementou-se também uma opção para o utilizador repôr os filtros, mas onde não foi utilizada nenhuma UDF.

Os dados relativamente às UDFs estão no ficheiro UDFs.sql.

Normalização

A normalização é uma técnica para organizar informação num SGBD. É usada, principalmente, para eliminar informação redundante e para assegurar que as dependências existentes fazem sentido. Existem três tipos, sendo que cada uma tem menos dependências que a anterior:

- Primeira Forma Normal
- Segunda Forma Normal
- Terceira Forma Normal

Visto que os nossos diagramas se encontram na 3FN, não foi necessário fazer alterações de normalização.

CONCLUSÃO

Concluindo o projeto, podemos verificar que os Stored Procedures e User Defined Functions conseguem ser bastante mais eficazes e intuitivos de usar do que uma simples query. Com estes procedimentos e funções é possível aumentar a rapidez diminuindo repetições de código e aumentando a organização e perceção do mesmo. Este trabalho foi bastante importante para aprofundar os nossos conhecimentos relativamente à base de dados, bem como aprender a trabalhar com VisualBasic.Net.

Ao longo de todo o código tentamos implementar da melhor maneira soluções que pensamos ser viáveis utilizando os conhecimentos que já tínhamos e outros que fomos adquirindo ao longo do trabalho. Assim, não prometemos que este esteja bem otimizado, mas sim que tenha uma grande parte das funcionalidades bem implementadas.

BIBLIOGRAFIA

[1] Slides 2020/2021 da Unidade Curricular de Base de Dados

[2] W3 Schools SQL - https://www.w3schools.com/sql/

[3] Visual Basic Documentation - https://docs.microsoft.com/en-us/dotnet/visual-basic/