Base de Dados

Carlos Costa
Joaquim Sousa Pinto

Relatório Projeto Final de uma Imobiliária

D8 Imobiliárias

Diogo Cruz, 98595
Artur Romão, 98470
P5G5

DETI
Universidade de Aveiro
Entregue: 27-06-2021

ÍNDICE

1. Introdução

No âmbito da disciplina de Base de Dados foi-nos pedido para desenvolver um projeto que envolvesse a elaboração de um sistema de Base de Dados para um tema escolhido por nós, de problema real.

Foi escolhida a criação de uma Base de Dados e interface para uma Imobiliária que necessitava de serviços tanto para uso de clientes como para uso de trabalhadores da Imobiliária. Os clientes têm a possibilidade de listar imóveis e/ou adquirir aqueles que existem no mercado. Os trabalhadores da Imobiliária podem ver informações sobre imóveis que estão encarregues.

A construção deste sistema foi realizada em SQL e com uma aplicação Windows Forms em Visual Studio com a linguagem Visual Basic.

2. Análise de Requisitos

O foco desta base de dados está na implementação de uma plataforma online de listagem e compra de imóveis por parte dos utilizadores e também de acesso a informações por parte dos agentes da imobiliária. Para isso falámos com alguns possíveis clientes (familiares de membros do grupo) e foi criada uma análise de requisitos com as observações adquiridas.

Percebemos inicialmente que devia existir uma entidade **Pessoa** que pode ser um **Agente** da imobiliária ou então um futuro **Interessado**/**Proprietário**.

De realçar que os clientes só são adicionados à entidade **Proprietário** no momento que este lista um imóvel pela 1ª vez e adicionado à entidade **Interessado** quando faz uma marcação ou proposta pela 1ª vez. Também não é possível um **Agente** ser **Proprietário**/**Interessado**.

Atributos de uma **Pessoa**:

- **NIF**
- **Nome**
- **Data de nascimento**
- **Endereço**
- **Email**
- **Número de telemóvel**
- **Password**

Para além destes atributos, à Entidade **Proprietário** está sempre associado um **Agente** e para cada Agente está associado um **Departamento** onde trabalha.

Atributos do **Departamento:**

- **Número de Departamento**
- **Localização**

Uma **Pessoa** pode fazer uma **Marcação** para visitar um ímovel que exista no mercado, sendo que este guarda os seguintes dados:

- **Data da marcação**

Uma **Pessoa** pode fazer uma **Proposta** a um ou mais imóveis que estejam no mercado, sendo que esta guarda os seguintes dados:

- **Valor da proposta**
- **Código da proposta**

Um **Imóvel** no mercado listado por algum **Proprietário** e contém os seguintes dados:

- **Código do imóvel**
- **Preço**
- **Localização**
- **Ano de construção**
- **Área total**
- **Área útil**

Cada Imóvel tem um tipo de **Negócio** entre os disponíveis na base de dados (Venda, Aluguer ou Trespasse) que contém os seguintes dados:

- **Referência de negócio do imóvel**

Cada **Imóvel** pode ser do tipo **Comercial** ou **Habitacional,** cada um com os seus atributos específicos que os distinguem.

Os tipos de imóveis comerciais podem ser Escritórios, Lojas ou Armazéns e contém os seguintes dados:

- **Estacionamento**

Os tipos de imóveis habitacionais podem ser Apartamentos, Quintas, Moradias, Terrenos ou Quartos e contém os seguintes dados:

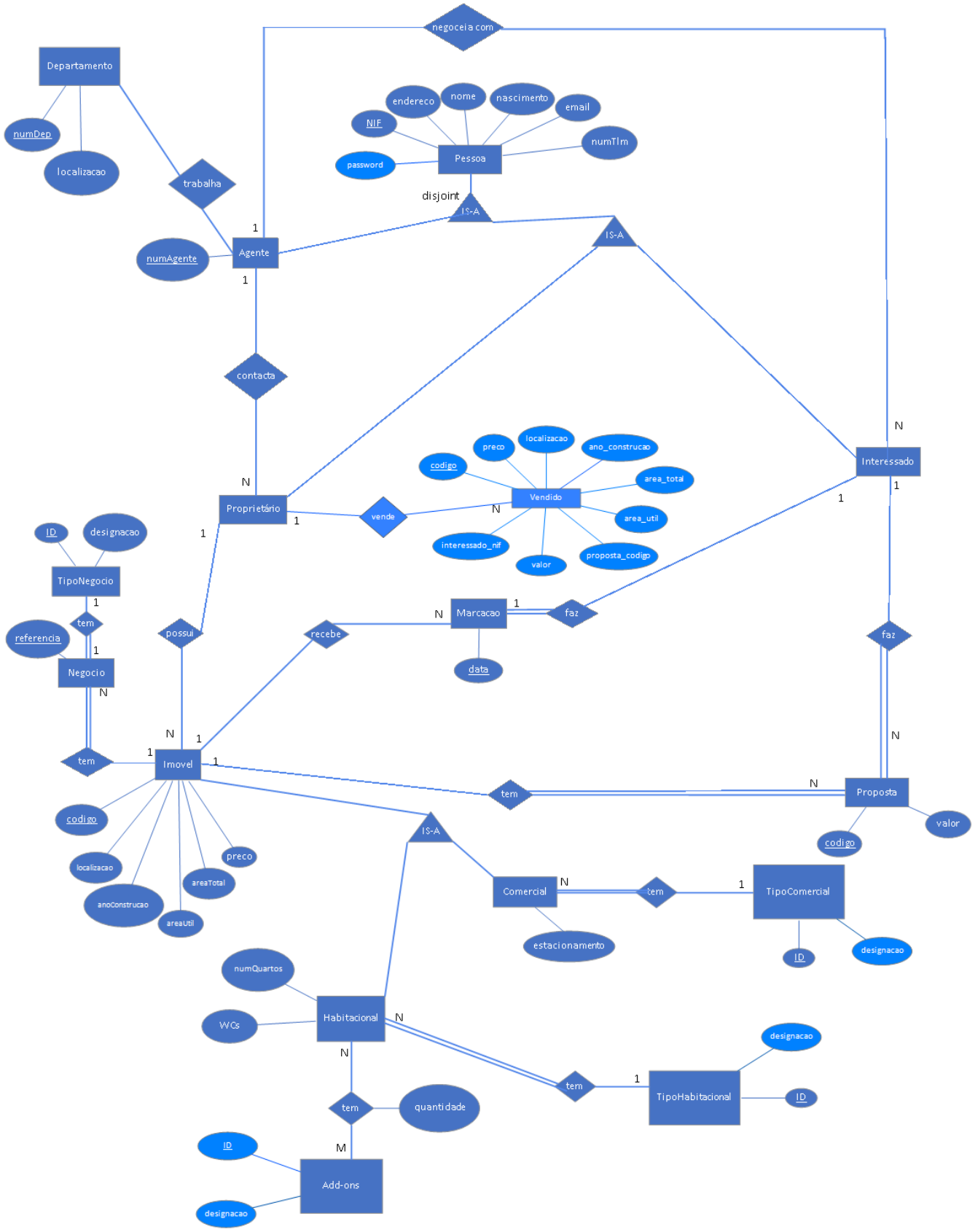
- **Número de quartos**
- **Número de WCs**

Os imóveis habitacionais também têm a possibilidade de ter **Add-Ons,** sendo que estes são características especiais que uma habitação pode ser. Exemplos destes podem ser churrasqueiras, piscinas, jardins, jacuzzis, garagem, painéis solares, etc.
Este contém os seguintes dados:

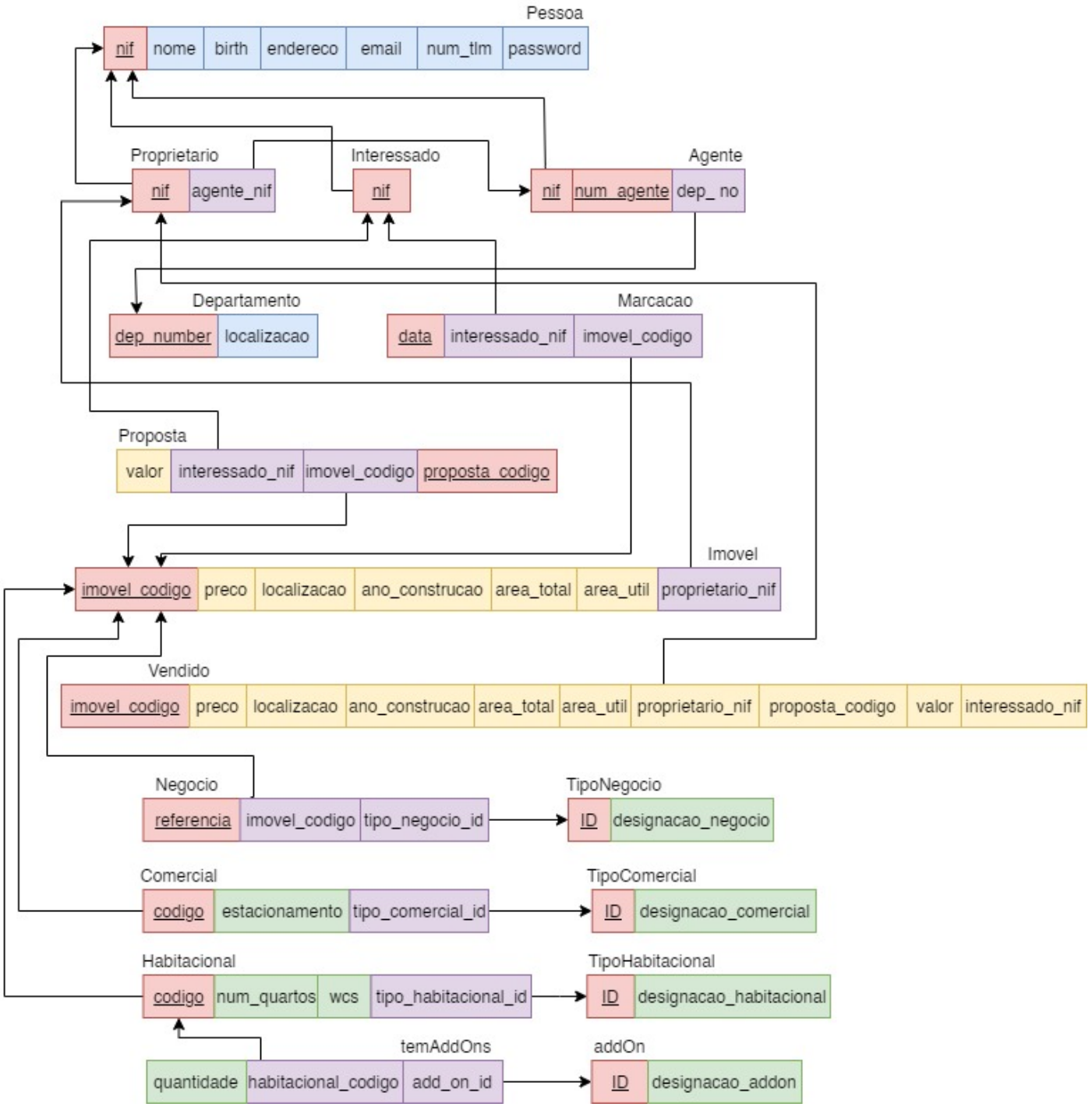
- **Quantidade**

Uma **Pessoa** pode, ao mesmo tempo, ser **Proprietária** de um ou mais imóveis e **Interessada** em um ou mais imóveis mas nunca ser mais que **Agente** da Imobiliária. Uma vez que o **Proprietário** aceita uma **Proposta** pelo seu **Imóvel** então este passa para o estado de **Vendido**, que contém as mesmas informações apresentadas na entidade **Imóvel** incluindo também a informação encontrada na entidade **Proposta**.

3. DER


negoceia com
Departamento
numDep
localizacao
trabalha
endereco
nome
nascimento
email
NIF
numTlm
password
Pessoa
disjoint
IS-A
IS-A
numAgente
Agente
contacta
Interessado
preco
localizacao
ano_construcao
codigo
area_total
Vendido
area_util
interessado_nif
valor
proposta_codigo
Proprietário
vende
ID
designacao
TipoNegocio
tem
referencia
Negocio
possui
recebe
Marcacao
faz
data
faz
tem
Imovel
tem
Proposta
codigo
valor
codigo
localizacao
anoConstrucao
areaTotal
areaUtil
preco
IS-A
Comercial
tem
TipoComercial
estacionamento
ID
designacao
numQuartos
WCs
Habitacional
tem
TipoHabitacional
designacao
ID
tem
quantidade
ID
Add-ons
designacao

4. ER


Pessoa
nif
nome
birth
endereco
email
num_tlm
password
Proprietario
nif
agente_nif
Interessado
nif
Agente
nif
num_agente
dep_ no
Departamento
dep_number
localizacao
Marcacao
data
interessado_nif
imovel_codigo
Proposta
valor
interessado_nif
imovel_codigo
proposta_codigo
Imovel
imovel_codigo
preco
localizacao
ano_construcao
area_total
area_util
proprietario_nif
Vendido
imovel_codigo
preco
localizacao
ano_construcao
area_total
area_util
proprietario_nif
proposta_codigo
valor
interessado_nif
Negocio
referencia
imovel_codigo
tipo_negocio_id
TipoNegocio
ID
designacao_negocio
Comercial
codigo
estacionamento
tipo_comercial_id
TipoComercial
ID
designacao_comercial
Habitacional
codigo
num_quartos
wcs
tipo_habitacional_id
TipoHabitacional
ID
designacao_habitacional
temAddOns
quantidade
habitacional_codigo
add_on_id
addOn
ID
designacao_addon

5. SQL DDL

```
use p5g5;
GO

CREATE SCHEMA Proj;
GO
```

Foi criado o SCHEMA ***Proj*** para agrupar as entidades criadas para a Base de Dados.
Desta forma todas as tabelas criadas têm como nome *Proj.[TableName]* ou *p5g5.Proj.[TableName]*

Fig 1: Criação do *SCHEMA*

Para a criação de tabelas foi seguido o padrão aprendido nas aulas e é igual para todas as tabelas. São definidos atributos, chaves primárias e chaves estrangeiras.

Neste exemplo são usadas *CONSTRAINTs* para validar dados na Base de Dados (exemplo: A área de um imóvel não pode ser menor que 0m²)

```
CREATE TABLE Proj.[imovel] ( -- imovel
    imovel_codigo VARCHAR(5) NOT NULL,
    preco INT NOT NULL CHECK (preco > 0),
    localizacao VARCHAR(50) NOT NULL,
    ano_construcao INT,
    area_total INT NOT NULL CHECK (area_total > 0),
    area_util INT,
    CONSTRAINT ck_area CHECK (area_util <= area_total AND area_util > 0),
    proprietario_nif INT NOT NULL,

    PRIMARY KEY(imovel_codigo),
    FOREIGN KEY(proprietario_nif) REFERENCES Proj.[proprietario](proprietario_nif)
);
```

Fig 2: Exemplo da criação de uma tabela

6. SQL DML

No DML os comandos foram executados dentro *Stored Procedures* , *UDFs* e *Triggers* de forma a criar uma abstração entre a interface e a Base de Dados, aumentando a segurança visto que são feitas verificações. De forma a poder avaliar o sistema já vem com entidades criadas usando ***EXEC*** das *SP's* e também ***INSERT INTO Proj.[TableName]***.

```
EXEC Proj.[cp_create_pessoa] 839684855, 'Therine Twiggins', '1973-02-22', 'ttwiggins0@blinklist.com', '9 Sheridan Parkway', 938325651, 'L6ue9s7'
EXEC Proj.[cp_create_pessoa] 631748029, 'Kettie Borrott', '1967-06-23', 'kborrott1@w3.org', '15156 Eliot Junction', 937261093, 'vOXkhL5bxG'
EXEC Proj.[cp_create_pessoa] 683133443, 'Eward Napper', '1998-12-08', 'enapper2@pinterest.com', '5 Towne Park', 930965532, 'ikJpQPhY'
EXEC Proj.[cp_create_pessoa] 372273797, 'Waneta Banks', '1991-09-25', 'wbanks3@hatena.ne.jp', '26583 Spohn Terrace', 914967839, '5YFLzW7os7J7'
EXEC Proj.[cp_create_pessoa] 309192268, 'Ronica Poff', '1968-06-14', 'rpoff4@weather.com', '47 Porter Junction', 936196630, 'VkNROvRQc'
EXEC Proj.[cp_create_pessoa] 963467845, 'Araldo Spens', '2001-03-08', 'aspens5@naver.com', '05785 Stang Drive', 938232741, 'pu7VSfonK'
EXEC Proj.[cp_create_pessoa] 194114527, 'Leda Kerwood', '1988-03-25', 'lkerwood6@google.cn', '32 Reindahl Avenue', 908295464, '6cNV6cC6'
EXEC Proj.[cp_create_pessoa] 231175500, 'Matthus Isted', '1984-07-15', 'misted7@techcrunch.com', '977 Spenser Drive', 991592893, 'ypIJyXOp'
EXEC Proj.[cp_create_pessoa] 859381020, 'Stanley Pedreschi', '1971-07-28', 'spedreschi8@dion.ne.jp', '94580 Elka Circle', 938135148, 'lK9A2amA02K'
EXEC Proj.[cp_create_pessoa] 373381498, 'Thekla Piwell', '1961-02-20', 'tpiwell9@wired.com', '65635 Fair Oaks Terrace', 930998370, 'a2xlOIXV'
EXEC Proj.[cp_create_pessoa] 408325564, 'Klemens Alden', '1992-09-02', 'kaldena@naver.com', '5 Fuller Junction', 974832478, 'j2B0pI'
```

Fig 3: Inserção de dados na Base de Dados usando ***SPs***

```
-- MARCACAO
INSERT INTO Proj.[marcacao] (data_marc, interessado_nif, imovel_codigo) values ('2022-12-22', 112133071, 'A9406');
INSERT INTO Proj.[marcacao] (data_marc, interessado_nif, imovel_codigo) values ('2021-10-12', 116872071, 'B1313');
INSERT INTO Proj.[marcacao] (data_marc, interessado_nif, imovel_codigo) values ('2021-11-27', 117003456, 'B1649');
INSERT INTO Proj.[marcacao] (data_marc, interessado_nif, imovel_codigo) values ('2021-07-14', 118394750, 'C6640');
INSERT INTO Proj.[marcacao] (data_marc, interessado_nif, imovel_codigo) values ('2021-07-12', 119650587, 'C9695');
INSERT INTO Proj.[marcacao] (data_marc, interessado_nif, imovel_codigo) values ('2021-08-05', 119768849, 'D3315');
INSERT INTO Proj.[marcacao] (data_marc, interessado_nif, imovel_codigo) values ('2021-09-14', 491839661, 'F5434');
INSERT INTO Proj.[marcacao] (data_marc, interessado_nif, imovel_codigo) values ('2021-08-20', 723466311, 'W8827');
INSERT INTO Proj.[marcacao] (data_marc, interessado_nif, imovel_codigo) values ('2021-07-17', 580727968, 'X2281');
INSERT INTO Proj.[marcacao] (data_marc, interessado_nif, imovel_codigo) values ('2021-07-11', 631724018, 'X3241');
INSERT INTO Proj.[marcacao] (data_marc, interessado_nif, imovel_codigo) values ('2021-09-24', 590156211, 'X6994');
INSERT INTO Proj.[marcacao] (data_marc, interessado_nif, imovel_codigo) values ('2021-10-09', 373747923, 'Y1382');
INSERT INTO Proj.[marcacao] (data_marc, interessado_nif, imovel_codigo) values ('2021-10-10', 373747923, 'Y4414');
INSERT INTO Proj.[marcacao] (data_marc, interessado_nif, imovel_codigo) values ('2021-10-11', 590156211, 'Y7365');
```

Fig 4: Inserção de dados na Base de dados usando ***INSERT INTO***

7. Normalização

Tendo em conta o desenvolvimento do projeto , de forma reduzir a redundância de dados foi executado o processo de normalização de dados, concluindo que se encontravam na Terceira Forma Normal (*3FN).* Sendo este o caso pudemos avançar com o desenvolvimento do projeto.

8. Triggers

Foram criados ***TRIGGERS INSTEAD OF INSERT*** para, ao fazer chamadas no DML, serem executadas verificações de dados para verificar que tudo está correto.

Neste exemplo ao fazer ***INSERT INTO Proj.[marcacao]*** o *Trigger* é executado e verifica se o utilizador inseriu uma data futura para fazer marcação ou se não existe outra visita no dia em que escolheu, se tudo correr bem, o utilizador é adicionado à tabela de **Interessados** e é feita uma **Marcação** para aquele dia no imóvel que escolheu.

```
CREATE TRIGGER Proj.[trigger_marcacao] ON Proj.[marcacao]
INSTEAD OF INSERT
AS
BEGIN
    DECLARE @data_marc DATE, @interessado_nif INT, @imovel_codigo VARCHAR(5), @now DATE
    SET @now = CAST(DATEADD(DAY, 1, GETDATE()) AS DATE)

    SELECT @data_marc=data_marc, @interessado_nif=interessado_nif, @imovel_codigo=imovel_codigo FROM INSERTED;
    -- checkar se data é valida
    BEGIN TRY
        IF NOT EXISTS(SELECT data_marc FROM p5g5.Proj.[marcacao] JOIN p5g5.Proj.[imovel] AS I ON I.imovel_codigo= @imovel_codigo WHERE data_marc=@data_marc)
        BEGIN
            IF @data_marc > @now -- verificar se a data é maior q agora
            BEGIN TRAN
                -- criar interessado se ele ainda n ta na tablea
                IF NOT EXISTS(SELECT interessado_nif FROM p5g5.Proj.[interessado] WHERE interessado_nif = @interessado_nif)
                    -- criar interessado
                    INSERT INTO p5g5.Proj.[interessado] VALUES (@interessado_nif)

                    INSERT INTO Proj.[marcacao](data_marc, interessado_nif, imovel_codigo)
                                VALUES (@data_marc, @interessado_nif, @imovel_codigo)
            COMMIT TRAN
        END
        ELSE
            ROLLBACK TRAN
            RAISERROR('Imovel ja tem proposta nesse dia...', 16, 1)
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION
        RAISERROR('Couldnt add proposta...', 16, 1)
    END CATCH
END
GO
```

Fig 5: Exemplo de um trigger quando é criada uma marcação num imóvel

Este trigger verifica se o imóvel que o proprietário quer adicionar **Add-Ons** existe e é do tipo **Habitacional.**

```
CREATE TRIGGER  Proj.[trigger_addon] ON Proj.[temAddOn]
INSTEAD OF INSERT
AS
BEGIN
    DECLARE @quantidade INT, @habitacional_codigo VARCHAR(5), @add_on_id INT
    SELECT @quantidade=quantidade, @habitacional_codigo=habitacional_codigo, @add_on_id=add_on_id FROM INSERTED

    IF EXISTS(SELECT I.imovel_codigo FROM p5g5.Proj.[imovel] AS I JOIN p5g5.Proj.[habitacional] AS H ON I.imovel_codigo=H.imovel_codigo
                WHERE I.imovel_codigo=@habitacional_codigo) -- se imovel existe

        INSERT INTO Proj.[temAddOn](quantidade, habitacional_codigo, add_on_id)
            VALUES(@quantidade, @habitacional_codigo, @add_on_id)
    ELSE
        RAISERROR('Couldnt add add on..', 16, 1)
END
GO
```

Fig 6: Trigger a verificar se o imóvel é habitacional ao ser adicionada uma feature

9. Stored Procedures

Foram criados ***Stored Procedures*** que são usados nas chamadas à Base de Dados e servem para adicionar elementos a tabelas, fazendo verificações ***IF... ELSE*** e auxiliando-se de ***Transações*** e chamadas a ***Functions*** . Temos exemplo como *cp_add_proposta, cp_create_pessoa, cp_add_imovel_habitacional, etc...*

Neste ***SP*** é feita a criação de uma pessoa, após ela executar o processo de registo, notar que apesar da ***Procedure*** receber a palavra passe do utilizador ela é guardada na tabela após fazer ***ENCRYPTBYPASSPHRASE*** que garante a segurança dos dados do utilizador pois a Base de Dados não sabe a sua palavra passe.

```
CREATE PROCEDURE Proj.[cp_create_pessoa]  -- criar pessoa
@nif INT, @nome VARCHAR(50), @birth DATE, @email VARCHAR(50), @morada VARCHAR(50), @num_tlm INT, @password VARCHAR(64)
AS
DECLARE @pwd VARBINARY(64)
SET @pwd = (SELECT ENCRYPTBYPASSPHRASE('**********', @password))
BEGIN TRAN
BEGIN TRY
    IF NOT EXISTS(SELECT nif  FROM p5g5.Proj.[pessoa] WHERE nif=@nif)   -- se n existe pessoa c esse nif, adciona
        INSERT INTO p5g5.Proj.[pessoa] VALUES (@nif, @nome, @birth, @morada, @email, @num_tlm, @pwd)
    COMMIT TRAN
END TRY
BEGIN CATCH
    IF @@TRANCOUNT > 0 ROLLBACK TRANSACTION
    RAISERROR('Couldnt create pessoa', 16, 1)
END CATCH
GO
```

Fig 7: Criação de um usuário, garantindo a segurança por parte da base de dados

Outro exemplo mais complexo é na criação de um **Imóvel** por parte de um **Proprietário**, neste caso, um imóvel **Habitacional.**

É verificado se o *NIF* existe na Base de Dados e se sim gera, com ajuda de *UDFs*, um **Agente** aleatório para trabalhar com o **Proprietário**, um código aleatório único para identificar o **Imóvel** (*imovel_codigo*) e um código aleatório para a ***Referencia*** *(referencia)* do **Negócio** referente ao **Imóvel**.

Depois destes dados criados, dentro de uma ***TRANSACTION***, uma séria de ***INSERT INTO p5g5.Proj.[TableName]*** são feitos e onde são inseridas em várias tabelas os dados que são referentes a cada entidade.

```
CREATE PROCEDURE Proj.[cp_add_imovel_habitacional] -- criar e adicionar a imovel, habitacional, proprietario e negocio
    @preco INT, @localizacao VARCHAR(50), @ano_construcao INT, @area_total INT, @area_util INT, @proprietario_nif INT,
    @num_quartos INT, @wcs INT, @habitacional_id INT, @negocio_id INT
AS
BEGIN
BEGIN TRANSACTION
BEGIN TRY
    SET NOCOUNT ON
    IF EXISTS(SELECT nif FROM p5g5.Proj.[pessoa] WHERE nif=@proprietario_nif) -- se o nif existe
    BEGIN
        -- se o nif_prop n existe
        IF NOT EXISTS(SELECT proprietario_nif FROM p5g5.Proj.[proprietario] WHERE proprietario_nif=@proprietario_nif)
        BEGIN
            DECLARE @agente_nif INT -- gerar agente aleatorio entre os que existem
            SET @agente_nif = (SELECT TOP 1 agente_nif FROM p5g5.Proj.[agente] ORDER BY NEWID())

            -- se ainda nao está na tabela de proprietarios, adiciona
            INSERT INTO p5g5.Proj.[proprietario] VALUES (@proprietario_nif, @agente_nif)
        END

        DECLARE @imovel_codigo VARCHAR(5) -- geração de um codigo random
        SET @imovel_codigo = (SELECT p5g5.Proj.[udf_createImovelCode]())

        DECLARE @referencia VARCHAR(9) -- gerar referencia para negocio
        SET @referencia = (SELECT p5g5.Proj.[udf_createRefCode]())

        -- se nao for um imovel repetido, adiciona a tabela imovel
        IF NOT EXISTS(SELECT localizacao FROM p5g5.Proj.[imovel] WHERE localizacao=@localizacao)
            INSERT INTO p5g5.Proj.[imovel] (imovel_codigo, preco, localizacao, ano_construcao, area_total, area_util, proprietario_nif)
            VALUES(@imovel_codigo, @preco, @localizacao, @ano_construcao, @area_total, @area_util, @proprietario_nif)

        -- add tabela habitacional
        INSERT INTO p5g5.Proj.[habitacional] (imovel_codigo, num_quartos, wcs, tipo_habitacional_id)
        VALUES(@imovel_codigo, @num_quartos, @wcs, @habitacional_id)

        -- add tabela negocio
        INSERT INTO p5g5.Proj.[negocio] (referencia, imovel_codigo, tipo_negocio_id)
        VALUES(@referencia, @imovel_codigo, @negocio_id)
        END
        COMMIT TRANSACTION
END TRY
BEGIN CATCH
    IF @@TRANCOUNT > 0 ROLLBACK TRANSACTION
    RAISERROR('Couldnt add imovel habitacional', 16, 1)
END CATCH
END
GO
```

Fig 8: Exemplo de uma ***SP*** que cria um imóvel habitacional

10. Views

Foram usadas ***Views*** para executar queries simples de busca de dados e também para a geração de strings aleatórias para criar os códigos que são usados para identificar os Imóveis, as Propostas e os Negócios

```
CREATE VIEW Proj.[view_imoveisVendidos]
AS
    SELECT V.imovel_codigo, V.preco, V.localizacao, V.ano_construcao, V.area_total,
                V.area_util, V.proprietario_nif, P.interessado_nif, P.valor
    FROM Proj.[vendido] AS V JOIN Proj.[proposta] AS P ON V.proposta_codigo = P.proposta_codigo
GO
```

Fig 9: Exemplo de uma ***View*** que retorna uma lista com os imóveis vendidos

```
-- criar codigo de imovel
CREATE VIEW Proj.[view_getImobRand]
AS
SELECT (CONCAT (CHAR(CAST((90-65)*RAND() + 65 AS INTEREGER)), FLOOR(RAND()*(10000-1000)+1000))) AS Value
```

Fig 10: ***View*** para gerar um código de imóvel aleatório (1 letra maiúscula + 4 números)

11. User Defined Functions

Em complementação com as ***Views*** foram usadas ***UDFs*** para executar *queries* simples de retorno de valores ou tabelas, mas em vantagem às ***Views*** aqui temos a possibilidade de usar parâmetros para especificar as procuras. Uma vez também que criámos as ***Views*** que geram códigos aleatórios, aqui temos as verificações que são gerados códigos garantidamente não repetidos. São usadas funções *Table-valued* e *Scalar-valued.*

```
-- get marcacoes para imovel
-- DROP FUNCTION Proj.[udf_getMarcacao]
-- GO

CREATE FUNCTION Proj.[udf_getMarcacao] (@cod VARCHAR(5)) RETURNS TABLE
AS
    RETURN (SELECT I.imovel_codigo, I.localizacao, I.proprietario_nif, M.data_marc
            FROM Proj.[imovel] AS I JOIN Proj.[marcacao] AS M ON I.imovel_codigo = M.imovel_codigo
            WHERE I.imovel_codigo = @cod)
GO
```

Fig 11: Função para ir buscar as marcações de um imóvel inseridos

```
CREATE FUNCTION Proj.[udf_createImovelCode] () RETURNS VARCHAR(5)
AS
BEGIN
    DECLARE @temp VARCHAR(5)
    SET @temp = (SELECT Value FROM p5g5.Proj.[view_getImobRand])

    WHILE EXISTS (SELECT imovel_codigo FROM p5g5.Proj.[imovel] WHERE imovel_codigo = @temp)
        BEGIN
            SET @temp = (SELECT Value FROM p5g5.Proj.[view_getImobRand])
        END
    RETURN @temp
END
GO
```

Fig 12: Função que valida o código aleatório gerado se não for repetido

12. Transactions

Foram usadas ***Transactions*** para garantir a consistência dos dados gerados na base de dados, garantindo que, mesmo que alguma das execuções de modificação de dados falhar, as outras operações também são revertidas. Estas transações foram usadas em casos onde há a alteração de vários dados envolvendo entidades diferentes e necessita-se que haja estabilidade entre todas.

```
CREATE PROCEDURE Proj.[cp_add_imovel_comercial] -- criar e adicionar a imovel, comercial, proprietario e negocio
    @preco INT, @localizacao VARCHAR(50), @ano_construcao INT, @area_total INT, @area_util INT, @proprietario_nif INT,
    @estacionamento BIT, @comercial_id INT, @negocio_id INT
AS
BEGIN
BEGIN TRANSACTION
BEGIN TRY
    SET NOCOUNT ON
    IF EXISTS(SELECT nif FROM p5g5.Proj.[pessoa] WHERE nif=@proprietario_nif) -- se o nif existe
    BEGIN
        -- se o nif_prop n existe
        IF NOT EXISTS(SELECT proprietario_nif FROM p5g5.Proj.[proprietario] WHERE proprietario_nif=@proprietario_nif)
        BEGIN
            DECLARE @agente_nif INT -- gerar agente aleatorio entre os que existem
            SET @agente_nif = (SELECT TOP 1 agente_nif FROM p5g5.Proj.[agente] ORDER BY NEWID())

            -- se ainda nao está na tabela de proprietarios, adiciona
            INSERT INTO p5g5.Proj.[proprietario] VALUES (@proprietario_nif, @agente_nif)
        END

        DECLARE @imovel_codigo VARCHAR(5) -- geração de um codigo random
        SET @imovel_codigo = (SELECT p5g5.Proj.[udf_createImovelCode]())

        DECLARE @referencia VARCHAR(9) -- gerar referencia para negocio
        SET @referencia = (SELECT p5g5.Proj.[udf_createRefCode]())

        -- se nao for um imovel repetido, adiciona a tabela imovel
        IF NOT EXISTS(SELECT localizacao FROM p5g5.Proj.[imovel] WHERE localizacao=@localizacao)
            INSERT INTO p5g5.Proj.[imovel] (imovel_codigo, preco, localizacao, ano_construcao, area_total, area_util, proprietario_nif)
            VALUES(@imovel_codigo, @preco, @localizacao, @ano_construcao, @area_total, @area_util, @proprietario_nif)

        -- add comercial
        INSERT INTO p5g5.Proj.comercial (imovel_codigo, estacionamento, tipo_comercial_id)
        VALUES(@imovel_codigo, @estacionamento, @comercial_id)

        -- add tabela negocio
        INSERT INTO p5g5.Proj.[negocio] (referencia, imovel_codigo, tipo_negocio_id)
        VALUES(@referencia, @imovel_codigo, @negocio_id)
    END
    COMMIT TRANSACTION
END TRY
BEGIN CATCH
    IF @@TRANCOUNT > 0 ROLLBACK TRANSACTION
    RAISERROR('Couldnt add imovel comercial', 16, 1)
END CATCH
END
GO
```

Fig 13: Exemplo de uma transaction para garantir que todos ou nenhuns dados são inseridos

13. Conclusão

Neste projeto não foram usados cursores nem indexes pois não achámos serem elementos necessários, compensando nos queries existentes.

Achamos que os objetivos do trabalho foram realizados apesar de acharmos que dedicámos demasiado tempo em *SQL Programming* sendo que muito do que fizemos lá acabou por não ser usado, atrasando um pouco as nossas ambições para a parte da interface do projeto que, apesar de concluída, pode não mostrar todas as funcionalidades criadas em *SQL*.

Em suma, o trabalho desenvolvido expandiu o nosso conhecimento em relação ao tema de Base de Dados e também às linguagens complementares como *SQL* e *Visual Basic*, desenvolvendo assim a nossa capacidade de construção de pequenas aplicações que necessitem de uma Base de Dados..

14. Bibliografia

Material fornecido pelos docentes da disciplina

https://elearning.ua.pt/course/view.php?id=318