Projeto Base de Dados

YourFM

André Morais 93236
Vicente Barros 97787

Departamento de Eletrónica Telecomunicações e Informática

22/06/2021

Índice

Introdução

A ideia projeto final, surge da incapacidade de estar numa viagem de carro e estar "presos" ao facto de só se puder estar a assistir ao que está a ser transmitido em direto da própria rádio, sem a capacidade de poder voltar atrás na transmissão ou através de plataformas de *streaming* fundamentalmente de música e podcasts sem a interação humana que a rádio oferece. A transmissão deixaria de ser baseada em FM e passava a ser através de IP, possibilitando assim implementar as várias caraterísticas pretendidas. Algumas a destacar é a capacidade de dar *rewind* na transmissão atual, assistir a programas que ocorreram anteriormente.

Requisitos Funcionais

Entidade	Funcionalidade
Utilizador	• Assistir a shows gravados • Seguir estações de radio • Consultar/alterar dados pessoais • Enviar mensagens em real-time para a transmissão • Pesquisar por shows • Adicionar um programa a uma lista de reprodução • Pesquisar pelo catálogo da plataforma através de género e rádio • Efetuar registo e login
Radio	• Consultar/alterar dados da estação • Adicionar/Remover shows • Adicionar/Remover locutores
Locutor	• Atualizar o programa • Ler/Responder a mensagens de utilizadores • Editar calendário da estação

Entidades

- **User**– Cliente final da plataforma.
- **Locutor** - Tipo de utilizador que está diretamente associado a uma estação de rádio.
- **Lista_Reprodução** - Todo o Utilizador poderá organizar os seus programas em listas personalizadas.
- **Estacão** – Tipo de utilizador que tem permissão para fazer transmissões.
- **Programa –** Conjunto de episódios.
- **Episódio -** Parte de um programa (NOTA: uma transmissão é um episódio de um programa que todas as rádios têm chamado *Live).*
- **Categoria** - Conjunto de keywords que uma estação ou programa pode ter.
- **Região** - Uma Estação irá pertencer a uma dada região.
- **Classificaçãoo** - Indicador de idade.
- **Mensagem –** Texto escrito enviado por um User para um chat.

DER – Diagrama Entidade-Relação

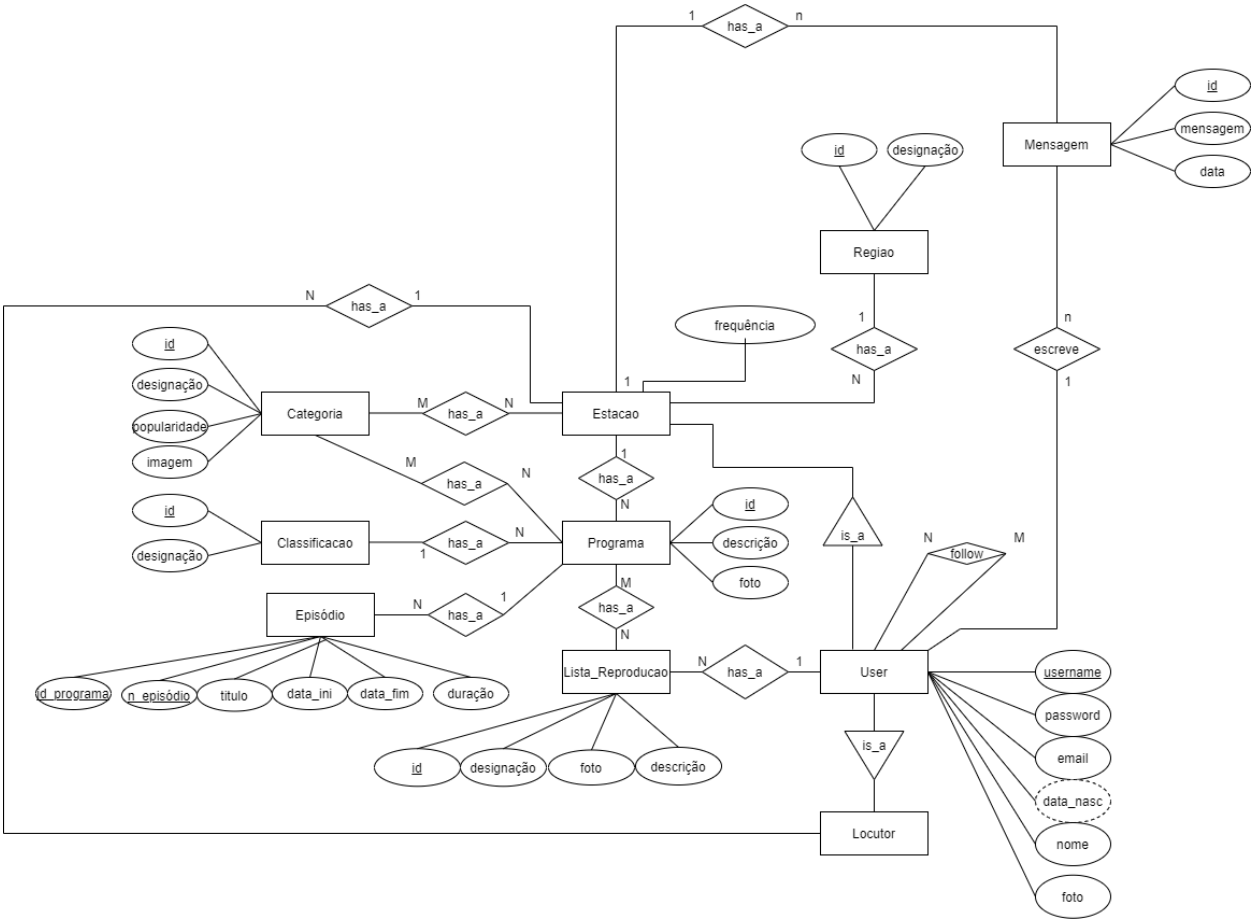

Figura 1-Diagrama DER

ER – Esquema Relacional

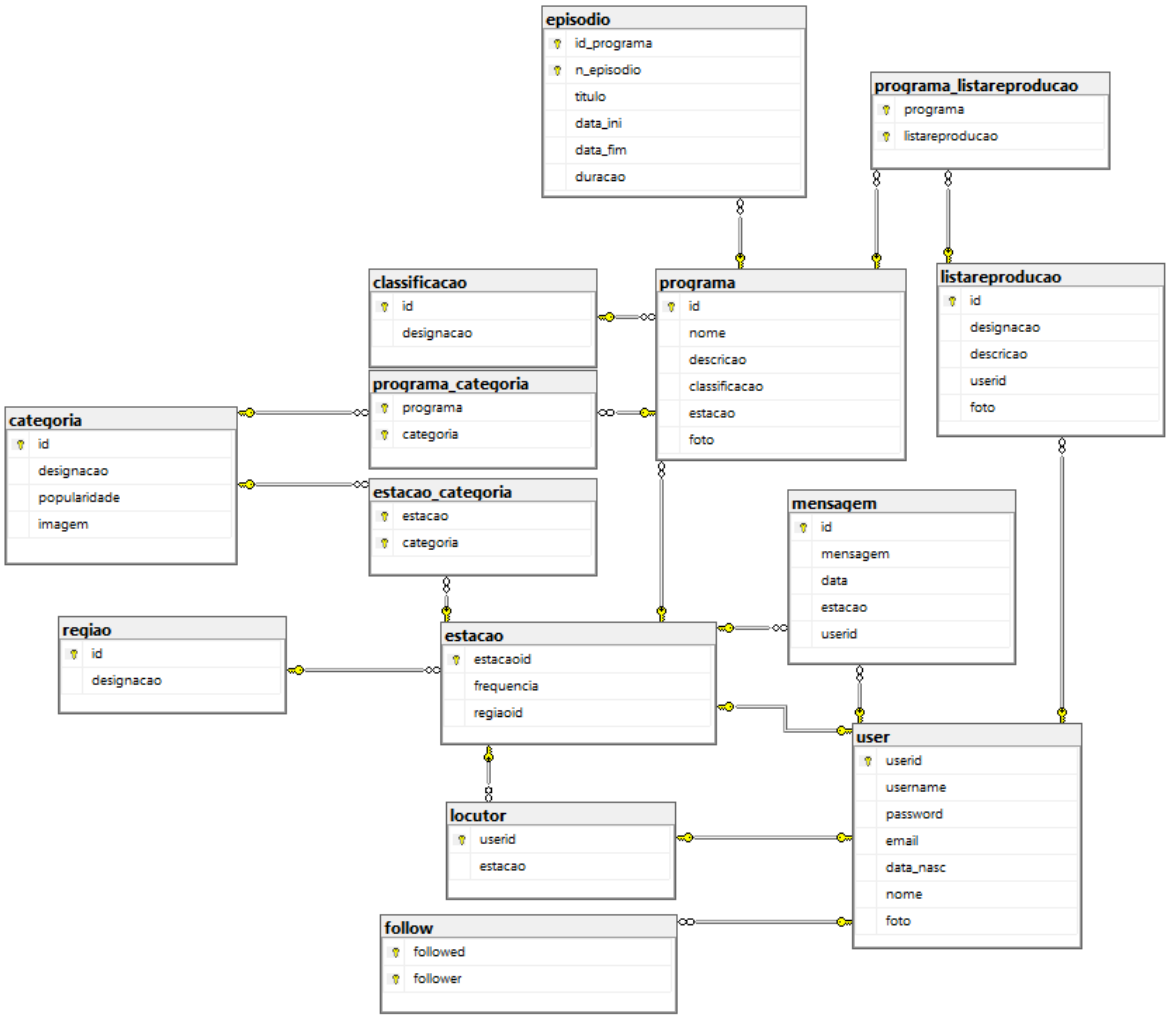

Figura 2-Diagrama ER

SQL DML

Todos os comandos de SQL DML foram executados dentro de *Stored Procedures*, *UDFs* e *Triggers* de maneira a criar uma camada de abstração entre a interface e a Base de Dados, aumentando a segurança com uma série de verificações, evitando *SQL Injections* e permitindo o encapsulamento da mesma.

Normalização

Tendo em conta as necessidades da plataforma foram definidas as relações e as entidades necessárias para o funcionamento da mesma. Durante todo o processo foram respeitadas as regras impostas com o intuito de obter a Terceira Forma Normal (3FN), respeitando as formas normais anteriores.

Índices

Ao analisar as consultas necessárias para o funcionamento da aplicação, concluiu-se que faria sentido utilizar um índice composto Non-clustered para melhorar a performance das pesquisas relativas às mensagens do chat. Sendo assim, foi criado o índice composto Non-clustered(data, estação) da tabela mensagem permitindo um aumento da performance a consultas necessárias para o funcionamento do chat.

Por outro lado, não foi necessário outro tipo de índices pelo simples facto que a maior parte das restantes consultas utilizem as Primary Keys para a pesquisa.

Triggers

Foi criado um Trigger After que irá ser ativado quando acontece a remoção de um dado programa da Base de dados, para remover todos os valores órfãos, isto é, todos os episódios desse programa, a entrada do dado programa nas listas de reprodução que o continham e as associações de categorias referentes a esse programa.

```
CREATE TRIGGER dropProgramasRelacoes ON dbo.programa
    AFTER DELETE
AS
BEGIN
    DECLARE @programaid AS INT
    SELECT @programaid = id FROM deleted;
    BEGIN TRY
        BEGIN TRAN T1
            DELETE FROM episodio WHERE id_programa = @programaid
            COMMIT TRAN T1

        BEGIN TRAN T2
            DELETE FROM programa_categoria WHERE programa = @programaid
            COMMIT TRAN T2
        BEGIN TRAN T3
            DELETE FROM programa_listareproducao WHERE programa = @programaid
            COMMIT TRAN T3

    END TRY
    BEGIN CATCH
        PRINT @@ERROR
        ROLLBACK TRAN T1
        ROLLBACK TRAN T2
        ROLLBACK TRAN T3
    END CATCH
END
```

Figura 3- Trigger dropProgramaRelações

Stored Procedures

Grande parte das chamadas à Base de Dados são feitas através de Stored Procedures para criar uma camada de abstração. Alguns *Stored Procedures* a destacar:

```
CREATE PROC dbo.doRegisto
    @username AS VARCHAR(18), @password AS VARCHAR(32), @email AS VARCHAR (
60), @data_nasc AS DATE, @nome AS VARCHAR(45), @foto AS varchar(256),
    @tipoUser AS VARCHAR(8) = "user"
AS

BEGIN
    DECLARE @userid AS INT
    DECLARE @data AS TABLE(id INT, utype VARCHAR(8))
    --Verificar se o username está em uso
    IF NOT EXISTS (SELECT username FROM [user] WHERE username=@username)
        BEGIN
            BEGIN TRAN T1
                BEGIN TRY
                    INSERT INTO [dbo].[user]
                        (username,[password], email,data_nasc,nome,foto)
                    VALUES
                        (@username, HASHBYTES('SHA2_256',@password),@email,
CAST(@data_nasc AS DATE),@nome,@foto);
                    COMMIT TRAN T1
                END TRY

                BEGIN CATCH
                    PRINT @@ERROR
                    ROLLBACK TRANSACTION T1
                    RETURN 0
                END CATCH;

            SET @userid = (SELECT userid FROM [user] WHERE
 username = @username)

            IF @tipoUser = 'station'
                BEGIN
                    BEGIN TRAN T2
                        BEGIN TRY
                            INSERT INTO estacao
                                (estacaoid,frequencia)
                            VALUES
                                (@userid,CAST((112+1000) AS DECIMAL(8,2)) /
10)
                            COMMIT TRAN T2
                        END TRY
                        BEGIN CATCH
                            PRINT @@ERROR
                            ROLLBACK TRANSACTION T2
                            RETURN 0
                        END CATCH
                END

            INSERT INTO @data VALUES (@userid,@tipoUser)
            SELECT * FROM @data
        END
    ELSE
        BEGIN
            PRINT 'Erro username em uso'
            RETURN 3
        END
END
```

Figura 4- ST responsável por lidar com o registo tanto de uma rádio como de um utilizador comum

```
CREATE PROCEDURE dbo.generoList @PageNumber AS INT
 = 1,@RowsPerPage AS INT = 16
AS
    SELECT
        c.designacao, c.imagem
    FROM dbo.categoria c
        RIGHT JOIN dbo.programa_categoria pc
            ON c.id = pc.categoria
        GROUP BY
 c.designacao, c.imagem,popularidade
        ORDER BY popularidade DESC

    OFFSET (@PageNumber-1)*@RowsPerPage ROWS
    FETCH NEXT @RowsPerPage ROWS ONLY
```

Figura 5-SP que permite a listagem de géneros que tenham pelo menos um programa através de paginação

```
CREATE PROC dbo.doLogin
    @username AS VARCHAR(18) = NULL,
    @password AS VARCHAR(32) = NULL
AS

BEGIN
    IF EXISTS (SELECT username FROM [user] WHERE (username=@username OR email = @username) AND
 [password] = HASHBYTES('SHA2_256',@password))
        BEGIN
            SELECT *
            FROM
            (
                        SELECT u.username, u.nome, u.foto, u.userid, 'locutor' as user_type
                        FROM [user] u
                            JOIN locutor l
                            ON l.userid = u.userid
                        WHERE u.username = @username
            UNION ALL
                        SELECT u.username, u.nome, u.foto, u.userid, 'station' as user_type
                        FROM [user] u
                            JOIN estacao e
                            ON e.estacaoid = u.userid
                        WHERE u.username = @username
            UNION ALL
                        SELECT u.username, u.nome, u.foto, u.userid, 'user' as user_type
                        FROM [user] u
                        WHERE u.username = @username
                        AND NOT EXISTS
                            (SELECT * from estacao e
                            WHERE u.userid = e.estacaoid
                            )
                        AND NOT EXISTS
                            (SELECT * from locutor l
                            WHERE u.userid = l.userid
                            )
            ) as myvalues
        END
    ELSE
        BEGIN
            RETURN 0;
        END
END
```

Figura 6-SP responsável pela validação do Login de um user

```
CREATE PROC dbo.detailsPrograma
    @id as int
AS
    SELECT
       p.nome,p.descricao,c.designacao as classificacao,p.foto,u.nome as estacao, AVG(ep.duracao) as
 duracao, COUNT(ep.id_programa) as numero_episodios
    FROM programa p
        INNER JOIN classificacao c
            ON p.classificacao = c.id
        INNER JOIN estacao e
            ON p.estacao = e.estacaoid
        INNER JOIN [user] u
            ON e.estacaoid = u.userid
        INNER JOIN episodio ep
            ON ep.id_programa = p.id
    WHERE p.id = @id
    GROUP BY p.nome,p.descricao,c.designacao,p.foto,u.nome
```

Figura 7-SP que permite a apresentação da informação de um dado programa

UDFs

As *User Defined Functions* foram usadas para a execução de consultas mais simples a nível de envio e processamento de dados, devido às suas restrições. Foi priorizado o uso de *UDFs* ao invés de *View* devido ao seu maior desempenho e a capacidade de passar parâmetros.

```
CREATE FUNCTION dbo.doSearch (@searchQuery AS VARCHAR(100) = '', @resultNumber AS INT) RETURNS
 @datatable TABLE (id INT ,nome VARCHAR(100), datatype VARCHAR(100))
AS
    BEGIN
        IF @searchQuery <> ''
            BEGIN
                --Procurar por programas
                INSERT @datatable SELECT TOP(@resultNumber) id,nome, 'programa' FROM programa WHERE nome
LIKE '%'+@searchQuery+'%'
                --Procurar por playlist
                INSERT @datatable SELECT TOP (@resultNumber) id,designacao,'playlist' FROM
 listareproducao WHERE designacao LIKE '%'+@searchQuery+'%'
                --Procurar por user
                INSERT @datatable SELECT TOP (@resultNumber) userid,username,'user' FROM [user] WHERE
 username LIKE '%'+@searchQuery+'%'
                --Procurar por Géneros
                INSERT @datatable SELECT TOP (@resultNumber) id,designacao,'genero' FROM categoria WHERE
 designacao LIKE '%'+@searchQuery+'%' ORDER BY popularidade
            END

        RETURN
    END
GO

SELECT * FROM dbo.doSearch('as',4);

SELECT * FROM dbo.doSearch('as',4) WHERE datatype = 'user';
```

Figura 8-UDF de pesquisa por termo

```
CREATE FUNCTION dbo.checkFollow (@followedUserID INT, @followerUserID INT) RETURNS INT
AS
BEGIN
    DECLARE @followStatus AS INT

    IF EXISTS (SELECT follower FROM dbo.follow WHERE followed = @followedUserID AND
 follower = @followerUserID)
        BEGIN
            SET @followStatus = 1
        END
    ELSE
        BEGIN
            SET @followStatus = 0
        END
    RETURN @followStatus
END
```

Figura 9-UDF para verificar se um user segue outro user

Transações

Para garantir a consistência dos dados, foram utilizadas transações, para caso aconteça alguma falha durante a modificação ou inserção de dados, a consistência dos dados não seja afetada.

```
CREATE PROC dbo.addEpisodio
    @id_programa AS INT, @titulo AS VARCHAR(50), @data_ini AS VARCHAR(80),@data_fim AS VARCHAR(80)
AS
BEGIN
    BEGIN TRAN
        BEGIN TRY
            INSERT INTO
                episodio (id_programa,n_episodio,titulo,data_ini,data_fim,duracao)
            VALUES
                (@id_programa,(SELECT MAX(n_episodio) FROM episodio WHERE id_programa=@id_programa)+1
,@titulo, CAST(@data_ini AS datetime), CAST(@data_fim AS datetime),DATEDIFF(hour,@data_ini,@data_fim))
        END TRY
        BEGIN CATCH
            RAISERROR('Erro ao inserir',16,1)
            PRINT @@ERROR
        END CATCH
    COMMIT TRAN
END
```

Figura 10- SP com uma Transação para lidar com a inserção de um dado episódio

Conclusão

Com este projeto pudemos ver na prática o funcionamento e a utilidade de um Sistema de Gestão de Base Dados, bem como a importância de ter um local centralizado da informação para permitir a consistência e segurança dos dados. Apesar de não ser a parte central do trabalho, o facto de ter uma interface intuitiva permitiu que a visualização dos dados e a interação com este fosse mais fluída.

Concluindo, os objetivos propostos no início do projeto foram em grande parte alcançados, permitindo assim o aprofundamento do conhecimento relacionado à matéria lecionada na cadeira de Base de Dados.