universidade de aveiro

BASE DE DADOS

Plataforma de Gestão de um *Private Torrent Tracker*

Trabalho Prático - P4G7

MESTRADO INTEGRADO EM ENGENHARIA COMPUTACIONAL

Álvaro Freixo, NMEC 93116

João Maria Machado, NMEC 89132

ÍNDICE

ÍNDICE **2**

INTRODUÇÃO **3**

MODELO DE DADOS **4**

Análise de Requisitos 4

Diagrama de Entidade Relação **5**

Modelo Relacional **7**

IMPLEMENTAÇÃO SQL **9**

Criação de Tabelas 9

Stored Procedures e Querys 10

Funções 11

Indexação 11

Cursores e Tabelas Temporárias 13

Views 14

Triggers 14

Transações 15

Conclusão **15**

BIBLIOGRAFIA **16**

INTRODUÇÃO

No presente trabalho pretende-se desenvolver um protótipo demonstrativo com inspiração numa plataforma de distribuição de *torrents* – um *torrent tracker* - meramente para fins educacionais.

O desenvolvimento do tema citado, possibilitou relacionar dados provenientes de realidades distintas, tal como num contexto real de conceção de uma base de dados. Desta forma, a aplicação de conceitos lecionados na unidade curricular é essencial. Utilização de *querys* complexas, implementação cuidadosa de *stored procedures* e a manipulação de dados foi uma constante.

A base de dados concebida é baseada num private *torrent tracker* real. Este trata-se de uma plataforma privada com uma comunidade coesa e fechada com regras próprias. Algumas das regras são referidas no documento de requisitos previamente entregue no primeiro momento de avaliação. Assim, o presente relatório subentende a leitura prévia do documento de requisitos. No entanto, as regras podem ser sumariadas no seguinte princípio: "...garantir a sobrevivência do conteúdo, a satisfação e a sustentabilidade da comunidade".

O presente relatório encontra-se dividido em 3 partes. Uma primeira parte na qual é explicada o modelo de dados dando destaques a algumas tabelas centrais ao projeto, uma segunda parte focada na introdução das tabelas e tuplos na base de dados e finalmente, uma terceira parte focada nos testes feitos à integridade do modelo e a sua validade.

Para além da análise de requisitos a leitura deste relatório pressupõem que o leitor tem acesso ao diagrama de entidade relação, ao esquema relacional, ao script que contém o DDL (*data definition language*), ao script que contém o DML (*data manipulation language*), ao *script* que contém a inserção dos tuplos para teste da BD e, finalmente, o acesso ao GUI desenvolvido pelo grupo que, por sua vez, implementa de maneira prática o trabalho desenvolvido.

A elaboração do presente projeto foi possibilitada pela utilização do programa *Microsoft SQL Sever Management Studio* e, para construção da Interface Gráfica (GUI), isto é desenhar os forms que o constituem, o programa *Microsoft Visual Studio* na linguagem C#.

O grupo considera que uma distribuição igualitária da nota obtida seria o mais indicado para a divisão de trabalho planeada.

MODELO DE DADOS

Análise de Requisitos

Antes de proceder à criação da base de dados propriamente dita foi necessário discutir os requisitos prévios à alocação dos dados. Para tal, procedeu-se à elaboração de uma lista de requisitos. Esta lista resulta de uma cuidada análise, discutida em grupo. Ao ter por base um site real, foi possível a enumeração e diferenciação dos tipos de dados necessários à criação de um diagrama de entidade-relação. Um dos aspetos a sublinhar é a criação de uma estrutura de dados capaz de suportar vários tipos de utilizadores com funções diferentes na comunidade, nomeadamente: "Utilizador Normal" capaz de descarregar conteúdo, utilizador conhecido como "*Uploader*" responsável pela inserção de novos conteúdos na plataforma,utilizador do tipo "*Staff*" capaz de moderar as interações na plataforma e recompensar utilizadores pelo bom comportamento e, finalmente, utilizador do tipo *Admin* com plenos poderes sobre a plataforma.

Outro aspeto que importa relevar é a criação de uma estrutura interna de organização de conteúdo. Para melhorar a organização da plataforma e oferecer ao utilizador uma experiência de pesquisa mais amigável os conteúdos são divididos em 3 categorias: Programas, Jogos, Conteúdo cuja informação é detalhada num site de crítica de cinema, que, por sua vez, se subdivide em séries e filmes.

Como forma de permitir a interação entre membros da comunidade e conteúdo implementou-se um sistema de comentários de modo que todos os utilizadores possam tecer comentários relativos a qualquer conteúdo disponibilizado na plataforma.

Finalmente, para incentivar o crescimento da comunidade, implementou-se um sistema de prémios segundo o qual, utilizadores podem ser recompensados pela boa conduta de interação com conteúdos ou até mesmo por recrutarem outros utilizadores.

Com estes requisitos considerados, procede-se à elaboração de um diagrama de entidade- relação.

Diagrama de Entidade Relação

O diagrama de entidade relação consiste na planificação da organização de informação na base de dados. Após todas as considerações feitas pela análise de requisitos é possível começar a planificar a estrutura dos dados bem como a maneira como estes se relacionam entre si.

Um processo importante, que acompanha a criação deste diagrama corresponde à escolha de chaves primárias para cada entidade incorporada no próprio diagrama. Uma chave primária é uma chave que identifica unicamente um tuplo numa tabela. Esta deve ser atribuída cuidadosamente, pois tem de ser imperativamente imutável, ou seja, não pode consistir numa caraterística que possa ser suscetível de modificação, caso contrário a integridade dos dados pode ser seriamente comprometida.

Um dos casos que mais importa mencionar é o caso do utilizador, no qual foi necessário criar um atributo próprio para atuar como chave primária – ID_User, é de notar que este atributo foi definido como um *integer* (*INT*) que é automaticamente incrementado à medida que os utilizadores vão aderindo à plataforma. Esta estratégia de escolha de chaves primárias foi utilizada em várias tabelas, sendo possível garantir a integridade dos dados na BD.

Uma outra situação que se revelou importante, a quando da criação do diagrama de entidade relação (e consequentemente durante todo o processo de criação da base de dados), foi o uso adequado de relações de hierarquias de classes (*Is-A*). Estas consistem numa classificação de dados que afirma que os dados classificados pertencem, hierarquicamente, a uma (ou várias) entidades posteriormente especificadas. Um exemplo desta implementação no nosso modelo de dados pode ser observado na figura 2

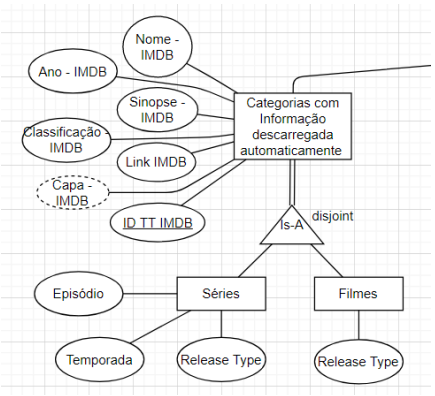

Figura 2.1: Exemplo de uma hierarquia de classes na implementação do diagrama entidade-relação. Neste caso é possível observar que uma categoria cuja informação é descarregada automaticamente é uma série ou filme.

Após considerar as propriedades previamente discutidas, foi possível elaborar um esquema de entidade-relação. Neste estão explicitados todos os dados que possam ser úteis na construção da base de dados de um *private torrent tracker,* bem como a maneira com estes se relacionam entre si (relações), a cardinalidade dos diferentes domínios dos atributos, explicitação das chaves primárias e atributos derivados, e, também, a obrigatoriedade que cada entidade tem (ou não) em participar numa relação.

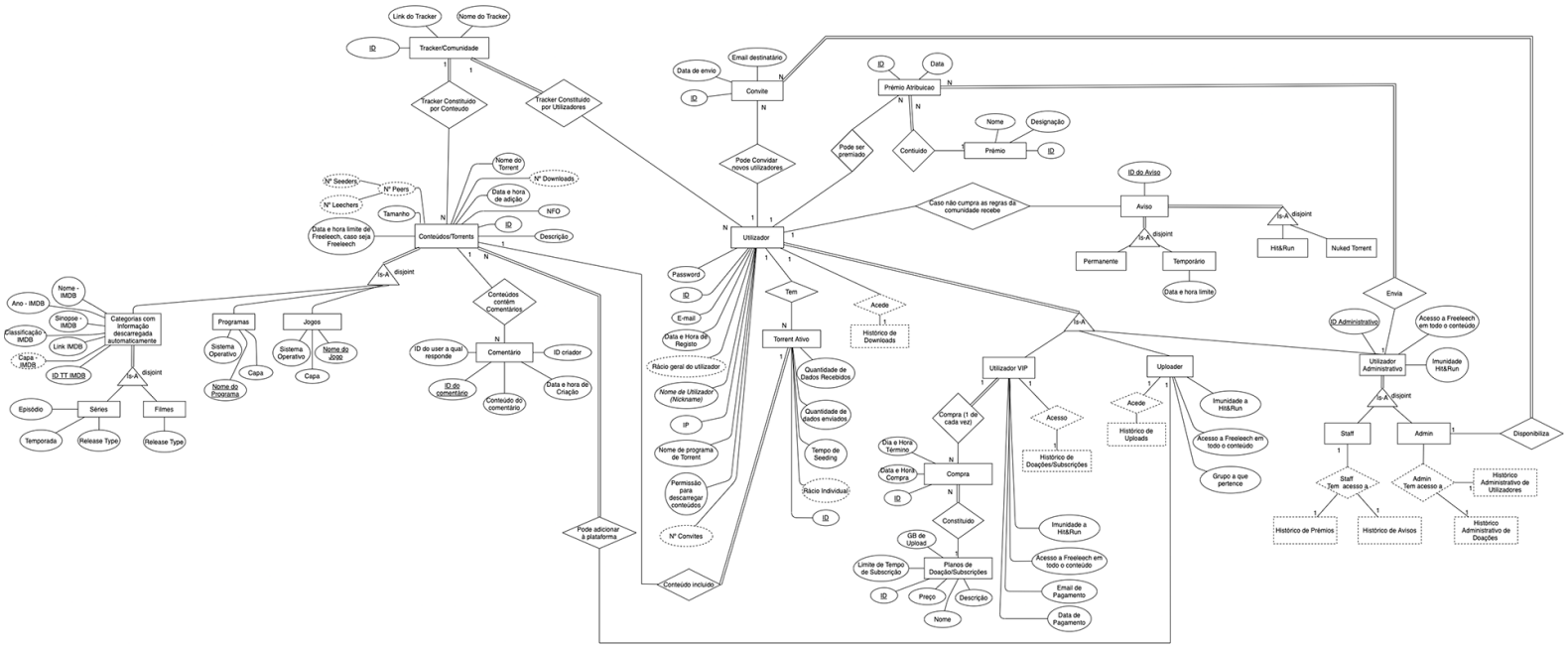

Figura 2.2: Imagem do diagrama entidade relação elaborado (Para uma melhor resolução recomenda-se a visualização da imagem em anexo).

Modelo Relacional

Após a criação do diagrama de entidade-relação procede-se à criação do modelo relacional. Contrariamente ao diagrama de entidade-relação, o modelo relacional consiste na visualização dos dados sob a forma de tabelas, de modo a diminuir o nível de abstração no processo de implementação e permitir a aproximação à linguagem SQL. Ou seja, a passagem do DER (diagrama entidade-relação) ao modelo relacional implica uma aproximação entre o que foi previamente discutido, a nível de requisitos, e a implementação que será realizada no DDL (*data definition language*).

No processo de transição do DER para o ER, aquilo que era previamente definido como atributo (ou até mesmo relação) será agora sob a forma de uma tabela na qual estão explícitos os seus atributos (também evidenciando a chave primária).

Nesta etapa ocorre, também, um processo extremamente importante no âmbito da questão da integridade dos dados, a definição de chaves estrangeiras. Uma chave estrangeira (Foreign key) consiste numa chave que é importada de outra tabela. A omissão, ou definição indevida de chaves estrangeiras pode facilmente originar conflitos de dados. A figura 2.2 mostra um exemplo da definição de chaves estrangeiras.

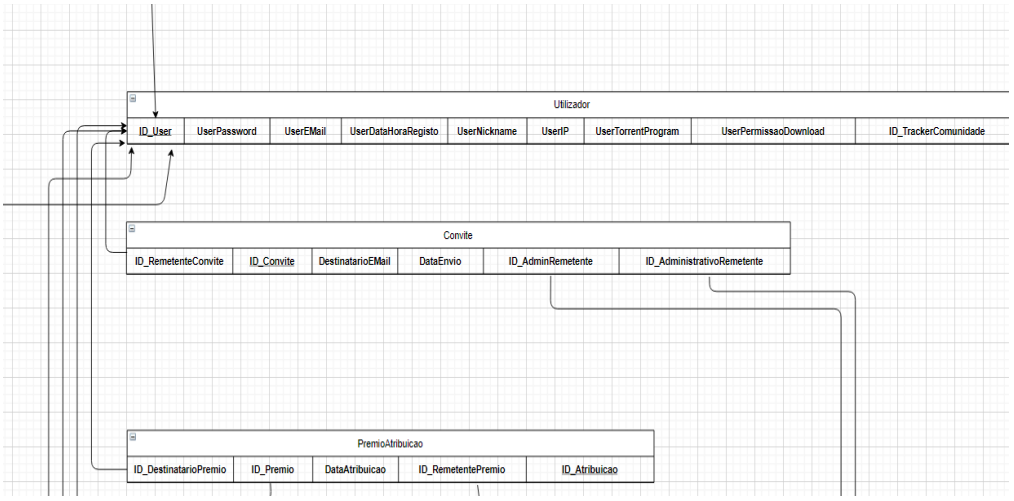

Figura 2.3: Exemplo de uma tabela cuja chave (*ID_User*) é chave estrangeira de 8 tabelas diferentes, o que é expectável, visto que várias entidades são moldadas pelo utilizador.

Uma última etapa a considerar, antes de prosseguir para a DDL, é a normalização de dados. Este processo tem apenas um propósito: a eliminação de redundâncias. Para o efeito, identificam-se potenciais dependências funcionais nas tabelas do ER, isto é, como dados numa tabela se relacionam entre si. Posteriormente classificam-se as dependências funcionais. Através da eliminação ordenada e sucessiva dos diferentes tipos de dependências funcionais podemos atingir a forma normal de Boyce-Codd (BCNF). Este algoritmo foi aplicado e verificado em todas as tabelas do nosso ER, desse modo evitando potenciais redundâncias. Um dos possíveis exemplos são as tabelas "PremioAtribuicao" e "Premio". O modelo relacional utilizado no projeto é evidenciado no arquivo "Modelo Relacional.png".

IMPLEMENTAÇÃO SQL

Criação de Tabelas

Nesta secção do relatório será explorado o processo de implementação do modelo de dados em SQL. Serão abordados alguns dos princípios que, durante a realização do trabalho, se revelaram determinantes.

Na secção anterior foi elaborado um esquema relacional, que nesta etapa será o principal guia para a criação de tabelas no SQL. O trecho de código 3.1 exemplifica a criação da tabela de utilizador.

```
CREATE TABLE TorrentTracker.Utilizador(
    ID_User                     TorrentTracker.ID           IDENTITY(1,1),
    UserPassword                VARCHAR(500)                NOT NULL,
    UserEMail                   VARCHAR(500)                NOT NULL,
    UserDataRegisto             SMALLDATETIME               NOT NULL,
    UserNickName                VARCHAR(256)                NOT NULL,
    UserIP                      VARCHAR(20),
    UserTorrentProgram          VARCHAR(256),
    UserPermissaoDownload       BINARY(1)                   NOT NULL,
    ID_TrackerComunidade        TorrentTracker.ID,

    PRIMARY KEY (ID_User)
)
GO
```

Figura 3.1: Exemplo de criação de uma tabela em SQL.

Desde já, uma das boas práticas implementadas foi o uso cuidadoso da declaração de variáveis a *"NOT NULL"*. Quando uma variável é declarada no SQL esta pode, preemptivamente, ser verificada como nunca ser nula. Ou seja, é requerido pelo modelo de dados que esta variável tenha um valor "existente". Este cuidado revela-se particularmente importante a quando da inserção de novos dados na base de dados pois determinados dados podem não existir e o modelo de dados pode continuar perfeitamente válido (aliás, em determinados casos é mesmo necessário que determinados atributos não existam).

Um aspeto que importa referir é o uso de variáveis declaradas com *"IDENTITY(1,1)"*. No decorrer da declaração de várias tabelas tem como atributo chave um ID. Este ID é único e imutável (chave primária), logo é conveniente que o modelo de dados a quando da sua criação o interprete como tal. Daí a utilidade de declarar este ID's com o atributo *"IDENTITY(1,1)"*. Ao criar as variáveis deste modo, garantimos que à medida que são introduzidos dados no modelo estes vêm sobre a forma de um inteiro que é incrementado.

Uma vantagem nativa ao SQL que também se provou útil é o facto de uma variável poder ser inicializada com o atributo "*SMALLDATETIME*" que atribui um valor de uma data seguida de uma hora a uma variável. Este tipo de variáveis é particularmente útil quando, por exemplo, tratamos de variáveis relacionadas com planos de subscrição que precisam de ser cuidadosamente calculadas para garantir que o utilizador tem acesso justo ao plano pelo qual pagou.

Após todas as considerações anteriores, foi possível fazer um pequeno teste à robustez do modelo de dados inserindo alguns tuplos gerados aleatoriamente com recurso ao site Generate Data. Geramos entre 200 (no máximo) a 5 (no mínimo) tuplos para cada tabela de modo aleatório.

Stored Procedures e Querys

Após a implementação do modelo de dados na secção anterior e a população na base de dados construída, cuidadosamente, foi analisado o tipo de acessos à BD para a tornar funcional. Grande parte destes acessos podem ser facilitados com o uso de stored procedures (SP's). Estes caracterizam-se por uma camada de abstração que permite o acesso à informação na BD.

Um dos *stored procedures* mais importantes, na elaboração deste trabalho, trata-se do *stored procedure* do *login*. Este procedimento é tão importante à nossa base de dados, pois é este que garante que a plataforma de *torrent tracking* é realmente privada (fechada). Na figura 3.2 podemos ver a implementação do stored procedure do *login* de um utilizador.

```
-- Login Page
CREATE PROCEDURE TorrentTracker.LoginProcedure @utilizador VARCHAR(256), @password VARCHAR(MAX)
  AS
  BEGIN
    SELECT * FROM TorrentTracker.Utilizador
    WHERE UserPassword=@password and UserNickName=@utilizador
  END
 GO
```

Figura 3.2 : *Stored procedure* implementado para o login de utilizador quando este deseja aceder à plataforma. Deve notar-se que para cada par de *inputs* utilizador e *password* a tabela de retorno deve ter no máximo uma linha e no mínimo zero. Caso contrário teríamos uma falha flagrante de segurança.

Um outro aspecto relevante, ainda a respeito do procedimento de login, foi o procedimento de segurança adotado. Isto é, para garantir que, se ocorrer algum problema de fuga de dados o utilizador não é comprometido. Para tal, a implementação de um sistema de encriptação de *password* é essencial.

É possível verificar as restantes *storage procedures* utilizadas nos ficheiros .sql anexados.

Funções

O uso de Funções como auxílio à construção de storage procedures foi fundamental para a reutilização e organização do código.

Na Figura 3.3 é possível verificar uma das mais usadas durante a construção de storages procedures.

```
CREATE FUNCTION TorrentTracker.User_NicknameToID (@user_nick VARCHAR(256)) RETURNS INT
AS
    BEGIN
        DECLARE @id_user INT;

        SET @id_user = (SELECT ID_User
                        FROM TorrentTracker.Utilizador
                        WHERE UserNickName = @user_nick);

        RETURN @id_user;
    END
GO
```

Figura 3.3: Função que converte o ID do utilizador no seu respectivo nickname, esta função foi particularmente importante a quando da implementação.

Indexação

A indexação permite a otimização na obtenção de queries (consultas definidas pelo administrador da BD) nos campos de pesquisa. Um dos locais (da interface gráfica) que será bastante utilizado caracteriza-se pela visualização e pesquisa dos conteúdos torrent em voga na plataforma. Na figura 3.4 está descrita a implementação no campo descrito. E na figura 3.5 a sua utilização na interface gráfica.

```
CREATE INDEX Index_ContentTorrent_Nome ON TorrentTracker.ContentTorrent(NomeContentTorrent);
GO
```

Figura 3.4: Exemplo da criação de um índice. Apesar de com os números de dados atuais não seja notório a necessidade de índice, se considerarmos que os utilizadores vão adicionando conteúdo à plataforma, o número de torrents crescerá e pesquisas podem tornar-se dispendiosas, daí a necessidade de índices.

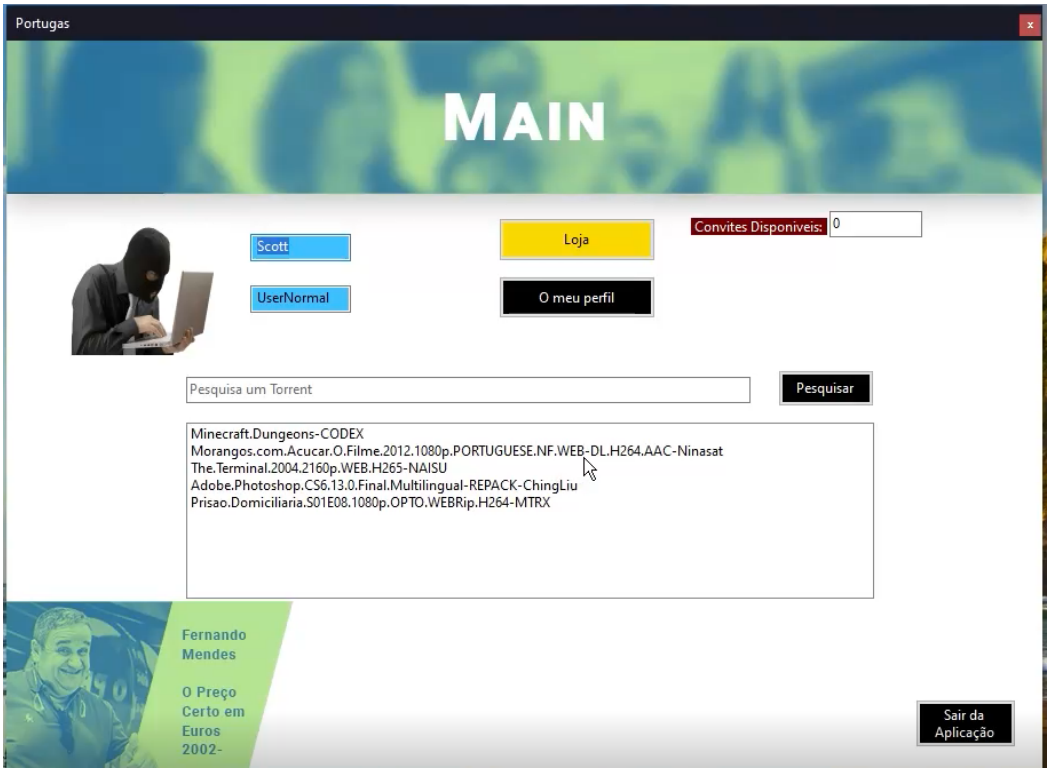

Figura 3.5: Exemplo de utilização de um índice para melhorar a pesquisa dos torrents disponível na plataforma.

Aliado ao bom funcionamento da indexação está a desfragmentação dos índices construidos pelo SGBD (SQL Server). A identificação da fragmentação das indexações definidas ficou a cargo da seguinte porção de código (Na figura 3.6).

```
SELECT s.[name] +'.'+t.[name]  AS table_name
 ,i.NAME AS index_name
 ,index_type_desc
 ,ROUND(avg_fragmentation_in_percent,2) AS avg_fragmentation_in_percent
 ,record_count AS table_record_count
FROM sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL, 'SAMPLED') ips
INNER JOIN sys.tables t on t.[object_id] = ips.[object_id]
INNER JOIN sys.schemas s on t.[schema_id] = s.[schema_id]
INNER JOIN sys.indexes i ON (ips.object_id = i.object_id) AND (ips.index_id = i.index_id)
ORDER BY avg_fragmentation_in_percent DESC
GO
```

Figura 3.6: Porção de código que permitiu a identificação dos índices presentes e a sua fragmentação.

Cursores e Tabelas Temporárias

As consultas a várias tabelas foram, também, auxiliadas pelo uso de cursores (percorrer tuplos das tabelas) e tabelas temporárias (armazenar temporariamente tuplos para utilização futura). Na figura 3.7 é possível verificar a sua utilização e, criação, na Figura 3.8.

```
ALTER PROCEDURE TorrentTracker.PremiosSpecificUser @user_nick VARCHAR(256)
 AS
    BEGIN
        DECLARE @id_user INT = (SELECT TorrentTracker.User_NicknameToID(@user_nick));
        DECLARE @id_atribuicao INT;
        DECLARE @id_remetente INT;

        IF NOT EXISTS (SELECT * FROM TorrentTracker.##TempPremios)
            DELETE FROM TorrentTracker.##TempPremios;


        DECLARE C CURSOR FAST_FORWARD
            FOR SELECT ID_Atribuicao, ID_RemetentePremio
                FROM TorrentTracker.PremioAtribuicao
                WHERE ID_DestinatarioPremio = @id_user;

        OPEN C;
        FETCH C INTO @id_atribuicao, @id_remetente;

        WHILE @@FETCH_STATUS = 0
            BEGIN
                INSERT INTO ##TempPremios ([ID],[Nome],[Design],[Dataa],[NomeE]) SELECT PA.ID_Atribuicao, Nome, Designacao, DataAtribuicao, NULL
                FROM (TorrentTracker.Premio AS P JOIN (TorrentTracker.PremioAtribuicao AS PA JOIN (TorrentTracker.Utilizador AS U JOIN TorrentTracker.UserAdministrativo AS UA
                                                                                                   ON U.ID_User = UA.ID_User)
                                                       ON PA.ID_RemetentePremio=UA.ID_Administrativo)
                                  ON P.ID_Premio=PA.ID_Premio)
                WHERE PA.ID_Atribuicao = @id_atribuicao;

                UPDATE ##TempPremios
                SET NomeE = (SELECT UserNickname
                             FROM TorrentTracker.Utilizador
                             WHERE ID_user=@id_remetente)
                WHERE ID = @id_atribuicao;

                FETCH C INTO @id_atribuicao, @id_remetente;
            END;
        CLOSE C;
        DEALLOCATE C;

        SELECT Nome,Design,Dataa,NomeE FROM ##TempPremios;
    END
GO
```

Figura 3.7: Exemplo de implementação de cursores e tabelas temporárias como auxílio na construção de storage procedures. No caso a "SP" de disponibiliza informação sobre os prémios específicos a cada utilizador.

```
ALTER PROCEDURE TorrentTracker.Make_Temp_Tables
AS
    BEGIN
        CREATE TABLE TorrentTracker.##TempAvisos(
            Nome  VARCHAR(30) PRIMARY KEY,
            Dataa   SMALLDATETIME
        );

        CREATE TABLE TorrentTracker.##TempPremios(
            ID INT PRIMARY KEY,
            Nome    VARCHAR(256),
            Design  VARCHAR(MAX),
            Dataa   SMALLDATETIME,
            NomeE   VARCHAR(256)
        );


    END
GO
```

Figura 3.8: Criação/Inicialização de tabelas temporárias após o Login do Utilizador.

Views

A utilização de views (vista) para além de possibilitar a apresentação de dados de consultas, permitiu, também, a reutilização de código estruturando queries mais complexas aquando do seu uso repetido, figura 3.9.

```
CREATE VIEW TorrentTracker.HistoricoComprasGeral_View AS
    SELECT U.UserNickname, P.Nome, P.PrecoPlano, C.DataHoraCompra, C.DataHoraTermino
    FROM (((TorrentTracker.Compra AS C JOIN TorrentTracker.UtilizadorVIP AS UV ON C.ID_Comprador=UV.ID_User)
        JOIN TorrentTracker.Utilizador AS U ON U.ID_User=UV.ID_User) JOIN TorrentTracker.PlanosDoacaoSubscricoes AS P ON C.ID_Plano=P.ID);
GO
```

Figura 3.9: Exemplo de utilização de views, no caso da visualização do histórico geral de compras que o Admin tem acesso.

Triggers

Uma das medidas mais importantes de integridade de dados foi a implementação de triggers. Estes foram utilizados em momentos de inserção e modificação de dados presentes na BD. Apesar da camada de abstração, este tipo de modificações têm de ser limitadas por regras que permitam o bom uso dos dados e a estabilidade da BD. Assim o uso de triggers alerta para a ocorrência destas situações e possibilita a tomada de medidas preventivas.

No exemplo seguinte, Figura 3.10 , o trigger impede o registo de um novo utilizador com um email já existente (na BD).

```
CREATE TRIGGER TorrentTracker.Trigger_Utilizador_Email ON TorrentTracker.Utilizador AFTER INSERT, UPDATE
AS
    BEGIN
        DECLARE @email_adicionado VARCHAR(500);

        SELECT @email_adicionado = UserEMail FROM inserted;

        IF (SELECT COUNT(*) FROM TorrentTracker.Utilizador WHERE UserEMail = @email_adicionado) > 1
            BEGIN
                RAISERROR('ERRO -> Outros utilizadores já tem esse email', 18, 1);
                ROLLBACK TRAN;
            END
    END
GO
```

Figura 3.10: Exemplo da criação de triggers.

Transações

A implementação de transações permitiu a modificação/inserção de informação em várias tabelas no mesmo storage procedure. Estas foram utilizadas em todas as ações de modificação que justificassem o seu uso, a fim de uma estruturação eficiente do código escrito. A figura 3.11 demonstra a sua implementação.

```
ALTER PROCEDURE TorrentTracker.ComprarPlano @user_nick VARCHAR(256), @plano_nome VARCHAR(256), @email_pagamento VARCHAR(500)
AS BEGIN
        BEGIN TRY
                DECLARE @lastseed_Compra INT = (SELECT COUNT(*) FROM TorrentTracker.Compra);
                DBCC CHECKIDENT('TorrentTracker.Compra', RESEED, @lastseed_Compra);

                DECLARE @id_user INT = (SELECT TorrentTracker.User_NicknameToID(@user_nick));
                DECLARE @plano_id INT = (SELECT TorrentTracker.User_PlanoNomeToID(@plano_nome));
                DECLARE @dias INT = (SELECT LimiteTempoPlano FROM TorrentTracker.PlanosDoacaoSubscricoes WHERE ID=@plano_id);
                DECLARE @data_atual SMALLDATETIME = CAST(GETDATE() AS smalldatetime);

                    BEGIN TRANSACTION
                        INSERT INTO TorrentTracker.UtilizadorVip([ID_User],[ImunidadeHitRun],[FreeleechTotal],[EMailPagamento],[DataHoraPagamento])
                        VALUES (@id_user, 1 ,1, @email_pagamento, @data_atual);

                        INSERT INTO TorrentTracker.Compra ([ID_Comprador],[ID_plano],[DataHoraCompra],[DataHoraTermino])
                        VALUES (@id_user, @plano_id, @data_atual, DATEADD(day, @dias, @data_atual));
                    COMMIT TRAN

                SELECT 'return Value' = 'Compra realizada com Sucesso!'
        END TRY
        BEGIN CATCH
                SELECT 'return Value' = 'ERRO -> O utilizador e/ou o plano não existem!'
        END CATCH
    END
GO
```

Figura 3.11: Exemplo de implementação de transações. No caso a transação é usada para preencher duas tabelas (UtilizadorVip, Compra) de modo que uma compra de um plano de subscrição seja concretizada.

Conclusão

O desenvolvimento do presente projeto possibilitou a obtenção de uma perceção mais extensiva da importância na necessidade de utilização de Bases de Dados no quotidiano. A crescente abundância de informação e urgente necessidade de organização e classificação desta é uma área em transformação e, cada vez mais, dotada de investimento.

Apesar, não somente, da complexidade do tema em questão, mas também do desejo de obtenção de um modelo realista, o grupo está satisfeito com o resultado obtido quer a nível de Interface Gráfica, quer na Base de Dados construída. Certamente que o conhecimento obtido será utilizado num futuro próximo.

BIBLIOGRAFIA

1. Ramez Elmasri and Shamkant B. Navathe. 2015. Fundamentals of Database Systems (7th. ed.). Pearson.
2. Carlos Costa. 2021. Slides de Base de Dados 2020/21, Universidade de Aveiro.
3. Website w3schools, w3schools.com.
4. Website tutorialgateway, tutorialgateway.org.
5. Website docs.microsoft.com.
6. Website stackoverflow.com.
7. Website social.msdn.microsoft.com.