universidade de aveiro

Bases de Dados
Relatório Final

Projecto:	Sistema de Distribuição de VideoJogos
Grupo:	P6G2
	Daniel Gomes (93015)
	Bruno Bastos (93302)
Disciplina:	Bases de Dados
Data de preparação:	11 de Junho de 2020

Introdução

No âmbito do trabalho final da disciplina de Bases de Dados, este relatório tem como principal objetivo apresentar o trabalho final pelo grupo **P6G2** tendo em conta, claramente, o tema escolhido para a realização deste.

O tema escolhido foi a modelação de um sistema de Distribuição de Videojogos, sendo o foco principal o desenho e desenvolvimento da camada de Base de Dados. Pretende-se assim guardar os diversos dados e registos das entidades constituintes do sistema como por exemplo clientes, jogos, compras efetuadas, plataformas e Empresas, entidades estas que irão ser referidas a seguir.

Foi realizada uma aplicação com interface gráfica do lado do Cliente e Administrador da plataforma desenvolvida em Windows Forms utilizando a linguagem de programação C#. Esta aplicação serve como uma camada de abstração para os dados que constituem a Base de Dados desenvolvida, possibilitando assim aos utilizadores desta aplicação, a realização de diversas operações.

Análise de Requisitos

Feita a Introdução, nesta secção irá-se abordar o objetivo concreto deste relatório, que como já foi enunciado, é a Análise de Requisitos. Nesta análise, procedeu-se à identificação das entidades presentes no Sistema, assim como as características e relações que estas estabelecem entre si.

Assim, encontra-se, de seguida a lista de requisitos:

- Um User (**User**) na plataforma pode ser um Cliente ou Administrador tem como atributos um UserID, Email, Data de registo e uma password.

- Cliente do sistema (**Client**) tem a capacidade de explorar e comprar jogos. Tem como atributos Username, Nome Completo, Data de nascimento, Imagem de Perfil, Sexo e Saldo.

- Um Cliente pode dar "follow" a outro para saber quais jogos esse joga. O outro Cliente não é obrigado a seguir de volta.

- O Administrador (**Admin**) é o responsável pela gestão e manutenção do sistema.

- A Empresa (**Company**) é responsável pelo desenvolvimento e publicação de jogos na plataforma. Esta tem como atributos ID, Nome de Empresa, website, logótipo, contacto, cidade, país e a data de Fundação.

- O Jogo (**Game**) é o elemento central da plataforma caracterizado, por um ID e nome único, data de lançamento, descrição , preço, restrição de Idade e uma imagem de Capa. Tem a possibilidade de pertencer a uma única Franchise.

- Um Género (**Genre**) representa a categoria a que pertence um jogo cujos atributos são um Nome único e uma descrição.

- Um jogo poderá pertencer a um ou mais géneros.

- Uma Plataforma (**Platform**), corresponde às Consolas (ou PC) na qual um jogo está disponível. Possui um nome único, data de lançamento no mercado e fabricante.

- Um jogo tem pode ter uma ou mais plataformas onde está disponível.

- Uma **Review**, tem como definição a avaliação de um jogo feita por um Utilizador. Contém um ID único, o ID do jogo e do cliente, título, texto relativo à opinião do utilizador , rating (de 0 a 5) e data da publicação da Review.

- Um Cliente pode partilhar a sua opinião sobre um Jogos através de uma Review. Só permitido ao cliente fazer uma review por jogo.

- Uma **Franchise** corresponde a uma série de *Games* que possuem mecânicas e história base semelhantes (Assassin's Creed, Call of Duty, FIFA, entre outros). Cada Franchise possui um ID e nome únicos e Logótipo.

- Uma Franchise pode ter vários jogos associados.

- Uma Empresa pode ter várias Franchises.

- Uma Cópia (**Copy**), corresponde a replicação do Jogo, tendo a si associada um Número de Série, o ID do Jogo e a plataforma para que a cópia do Jogo foi produzida.

- Um Jogo pode ter várias Cópias.

- Uma Compra (**Purchase**) corresponde a uma transação de um jogo por parte de um Cliente para uma dada plataforma. Tem como atributos um Número de Compra, preço associado, data da compra, UserID do Cliente que fez a compra e o ID do produto adquirido. Só poderá efetuar a compra de cada jogo para uma plataforma específica e não é possível comprar o mesmo jogo para plataformas diferentes.

- Cada Compra só pode ser efetuada caso haja saldo suficiente. Assim o utilizador tem a possibilidade de carregar a "carteira" com dinheiro.

- Carregamento (**Credit**) , corresponde a transferência de dinheiro para a conta da plataforma.Neste sistema, os jogos são comprados a partir do saldo do cliente, pelo que é necessário que este transfira dinheiro para a plataforma. Um carregamento tem um número, método de pagamento, ID do cliente e data de carregamento associados.

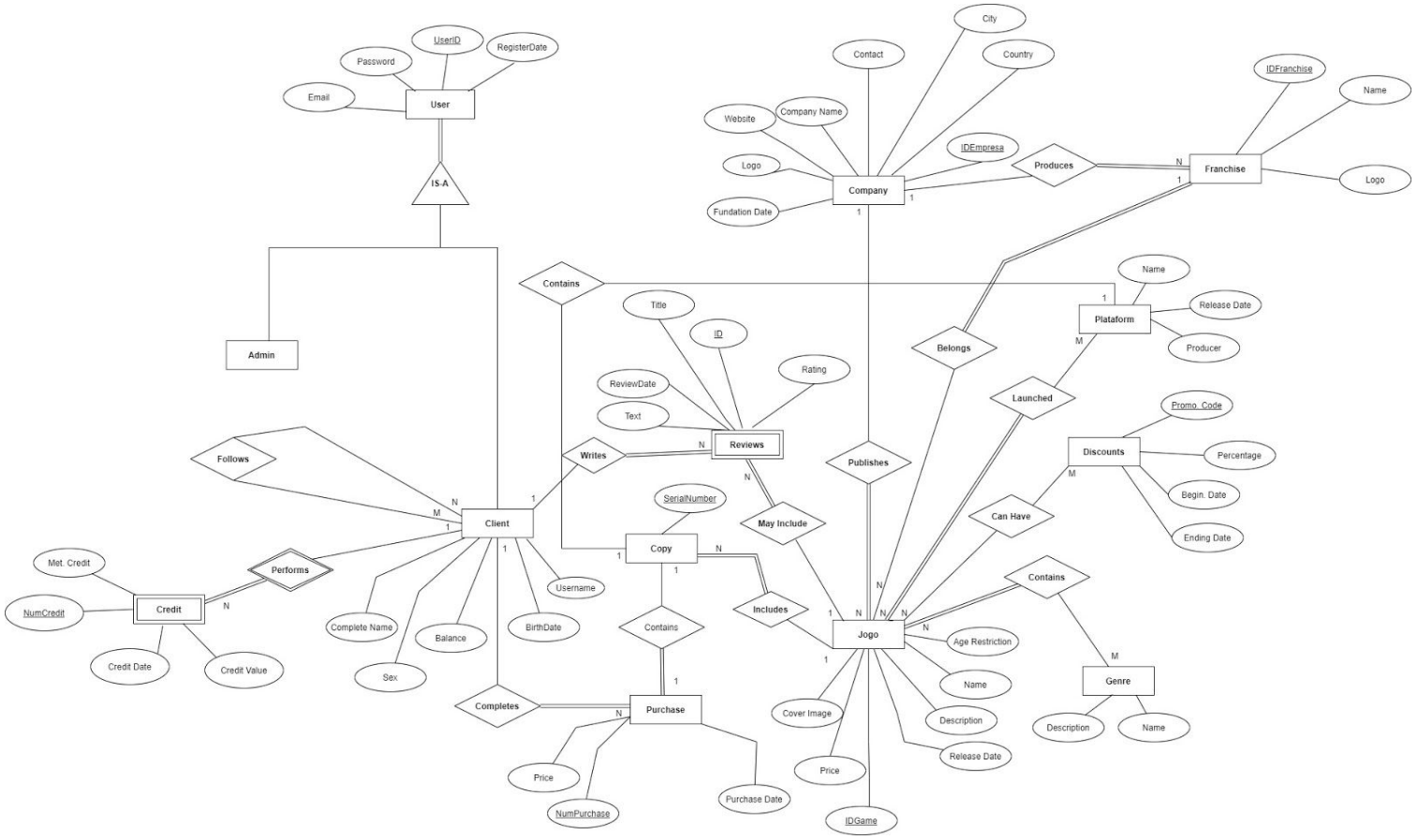
- Um Cliente pode fazer vários carregamentos.

- Um Desconto (**Discount**) é uma redução no preço base de um Jogo. Este é caracterizado por um código promocional, percentagem de redução e data de validade.

- Um Jogo pode ter um desconto associado, podendo apenas estar um desconto ativo para cada jogo num dado momento.

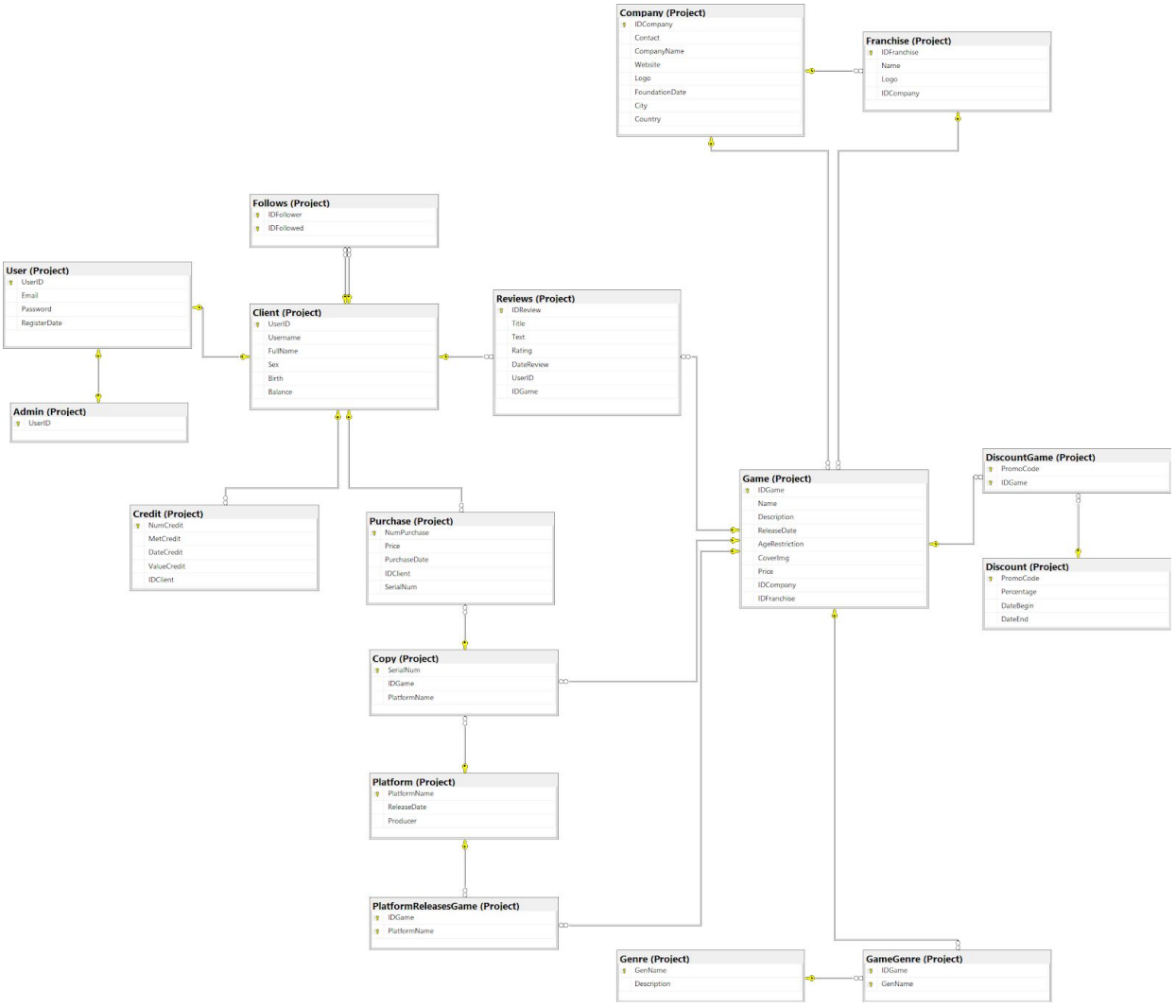
Diagrama Entidade Relação

Após feita a análise de Requisitos procedeu-se a realização do Diagrama Entidade Relação do nosso modelo de Dados.



Esquema Relacional -SQL Server

Na imagem seguinte, encontra-se aquela que é a versão final do esquema relacional do nosso Sistema, auto gerada pelo SQL-Server.



Normalização

Devido ao facto das nossas entidades não possuírem muitos atributos, não ocorreu a presença de Entidades não normalizadas. Concluímos que a nossa Base de Dados se encontrava na Terceira Forma Normal, visto que, por um lado, cumpria os requisitos da Primeira e Segunda Forma Normal, mas que, por outro lado, não cumpria os Requisitos para estar na Quarta Forma Normal.

Todos os atributos das entidades são atómicos, estando assim na 1ª FN.

Todos os atributos que não são chave dependem unicamente da chave da relação, pelo que se encontra na 2º FN.

Não se encontra na 4ºFN visto que ocorre a presença de dependências multivaloradas.

Data Definition Language - DDL

Após as tarefas referidas anteriormente, a próxima etapa constituiu na criação de tabelas tendo em conta o Esquema Relacional, assim como a colocação das restrições de integridade sobre os Dados. Para isso, utilizou-se SQL-DDL(*Data Definition Language*). Além disso todas as tabelas foram criadas segundo o **SCHEMA** *Project.*

As tabelas **Game**, **Company e Franchise** têm uma imagem. A abordagem que seguimos para guardar as imagens, foi guardar um atributo com o link onde se encontra a Imagem Online.

```
CREATE TABLE Project.Game(

    IDGame          INT             IDENTITY(1,1),
    Name            VARCHAR(50)     NOT NULL,
    Description     VARCHAR(MAX),
    ReleaseDate     DATE            NOT NULL,
    AgeRestriction  INT             NOT NUll,
    CoverImg        VARCHAR(MAX),
    Price           Decimal(5,2)   CHECK(Price >= 0)  NOT NULL,
    IDCompany       INT             NOT NULL,
    IDFranchise    INT,
    CONSTRAINT chk_AgeRestrict CHECK (AgeRestriction >= 1 AND AgeRestriction <= 18),
    PRIMARY KEY(IDGame)
);
```

Na imagem seguinte encontram-se as restrições de integridade usadas (Chaves Estrangeiras).

```
-- Foreign keys
ALTER TABLE Project.[Admin] ADD CONSTRAINT AdminID FOREIGN KEY (UserID) REFERENCES Project.[User](UserID);
ALTER TABLE Project.Credit  ADD CONSTRAINT CredClient FOREIGN KEY(IDClient) REFERENCES Project.Client(UserID);
ALTER TABLE Project.Reviews ADD CONSTRAINT RevClient FOREIGN KEY(UserID) REFERENCES Project.Client(UserID);
ALTER TABLE Project.Reviews ADD CONSTRAINT RevGame FOREIGN KEY(IDGame) REFERENCES Project.Game(IDGame);
ALTER TABLE Project.Purchase ADD CONSTRAINT ClientPurchase FOREIGN KEY(IDClient) REFERENCES Project.Client(UserID);
ALTER TABLE Project.Purchase ADD CONSTRAINT GamePurchase FOREIGN KEY(SerialNum) REFERENCES Project.[Copy](SerialNum);
ALTER TABLE Project.Client ADD CONSTRAINT ClientID FOREIGN KEY(UserID) REFERENCES Project.[User](UserID);
ALTER TABLE Project.Game ADD CONSTRAINT   IDCompanyGame  FOREIGN KEY (IDCompany) REFERENCES Project.Company(IDCompany);
ALTER TABLE Project.Game ADD CONSTRAINT   IDFranchiseGame  FOREIGN KEY (IDFranchise) REFERENCES Project.Franchise(IDFranchise);
ALTER TABLE Project.[Copy] ADD CONSTRAINT CopyGame FOREIGN KEY(IDGame) REFERENCES Project.Game(IDGame);
ALTER TABLE Project.[Copy] ADD CONSTRAINT CopyPlatform FOREIGN KEY(PlatformName) REFERENCES Project.[Platform](PlatformName);
ALTER TABLE Project.PlatformReleasesGame ADD CONSTRAINT LaunchedGame FOREIGN KEY(IDGame) REFERENCES Project.Game(IDGame);
ALTER TABLE Project.PlatformReleasesGame ADD CONSTRAINT PlatformGame FOREIGN KEY(PlatformName) REFERENCES Project.[Platform](PlatformName);
ALTER TABLE Project.DiscountGame ADD CONSTRAINT CodDiscount FOREIGN KEY (PromoCode) REFERENCES Project.Discount(PromoCode);
ALTER TABLE Project.DiscountGame ADD CONSTRAINT CodGame FOREIGN KEY (IDGame) REFERENCES Project.Game(IDGame);
ALTER TABLE Project.GameGenre ADD CONSTRAINT  GenGame FOREIGN KEY (IDGame) REFERENCES Project.Game(IDGame);
ALTER TABLE Project.GameGenre ADD CONSTRAINT  GenName FOREIGN KEY (GenName) REFERENCES Project.Genre(GenName);
ALTER TABLE Project.Franchise ADD CONSTRAINT  Comp FOREIGN KEY (IDCompany) REFERENCES Project.Company(IDCompany);
```

Data Manipulation Language - DML

A linguagem **SQL-DML** (*Data Manipulation Language*) foi onde assentou grande parte do nosso trabalho, pois nos permitiu realizar as mais diversas operações na Base de Dados, operações estas que irão ser enunciadas posteriormente neste relatório. De forma a tentar evitar SQL Injection, procedeu-se a utilização de SQL Parametrizado.

```
-- insert platform

INSERT INTO Project.[Platform](PlatformName,ReleaseDate,Producer) VALUES ('Xbox 360','2005-11-22', 'Microsoft' );
INSERT INTO Project.[Platform](PlatformName,ReleaseDate,Producer) VALUES ('Xbox One','2013-11-22', 'Microsoft' );
INSERT INTO Project.[Platform](PlatformName,ReleaseDate,Producer) VALUES ('Windows 7','2009-07-22', 'Microsoft' );
INSERT INTO Project.[Platform](PlatformName,ReleaseDate,Producer) VALUES ('Windows 10','2015-07-29', 'Microsoft' );
INSERT INTO Project.[Platform](PlatformName,ReleaseDate,Producer) VALUES ('PlayStation','1994-12-03', 'Sony' );
INSERT INTO Project.[Platform](PlatformName,ReleaseDate,Producer) VALUES ('PlayStation 2','2000-03-04', 'Sony' );
INSERT INTO Project.[Platform](PlatformName,ReleaseDate,Producer) VALUES ('PlayStation 3','2006-11-11', 'Sony' );
INSERT INTO Project.[Platform](PlatformName,ReleaseDate,Producer) VALUES ('PlayStation 4','2005-11-13', 'Sony' );
INSERT INTO Project.[Platform](PlatformName,ReleaseDate,Producer) VALUES ('PlayStation Portable','2004-12-12', 'Sony' );
INSERT INTO Project.[Platform](PlatformName,ReleaseDate,Producer) VALUES ('PlayStation Vita','2011-12-17', 'Microsoft' );
INSERT INTO Project.[Platform](PlatformName,ReleaseDate,Producer) VALUES ('Nintendo DS', '2004-12-02','Nintendo' );
INSERT INTO Project.[Platform](PlatformName,ReleaseDate,Producer) VALUES ('Nintendo 3DS', '2011-02-26', 'Nintendo');
INSERT INTO Project.[Platform](PlatformName,ReleaseDate,Producer) VALUES ('Wii', '2006-11-19', 'Nintendo' );
INSERT INTO Project.[Platform](PlatformName,ReleaseDate,Producer) VALUES ('Nintendo Switch', '2017-03-03', 'Nintendo');
INSERT INTO Project.[Platform](PlatformName,ReleaseDate,Producer) VALUES ('Nintendo 64', '1996-06-23','Nintendo' );
INSERT INTO Project.[Platform](PlatformName,ReleaseDate,Producer) VALUES ('GameCube', '2001-09-14','Nintendo');
INSERT INTO Project.[Platform](PlatformName,ReleaseDate,Producer) VALUES ('Wii U', '2012-11-18','Nintendo');
INSERT INTO Project.[Platform](PlatformName,ReleaseDate,Producer) VALUES ('MacOS', '2018-11-24', 'Apple');
INSERT INTO Project.[Platform](PlatformName,ReleaseDate,Producer) VALUES ('iOS', '2007-06-29','Apple');
INSERT INTO Project.[Platform](PlatformName,ReleaseDate,Producer) VALUES ('Android', '2006-09-28','Google');
INSERT INTO Project.[Platform](PlatformName,ReleaseDate,Producer) VALUES ('Linux', '1991-04-20', 'Linux');
INSERT INTO Project.[Platform](PlatformName,ReleaseDate,Producer) VALUES ('Sega Saturn', '1994-11-22','Sega');
INSERT INTO Project.[Platform](PlatformName,ReleaseDate,Producer) VALUES ('Dreamcast', '1998-11-27','Sega' );
```

Views

A utilização de SQL **Views** no nosso trabalho foi nula, visto que demos preferência a utilização de UDFs, isto devido ao facto de apresentar uma melhor performance que **Views** e os casos de utilização de UDFs, como iremos ver a seguir, mostraram-se mais adequados utilizando UDFs do que Views.

Stored Procedures

Os Stored Procedures foram utilizados maioritariamente com vista a criar uma camada de abstração na escrita de Dados na Base de Dados assim como a possibilidade de pesquisa de Dados. É importante referir que a utilização destes teve em vista consumir os dados da Aplicação de uma forma mais segura, evitando por isso, a utilização de queries *ad-hoc* nestes casos.

Os stored Procedures usados tiveram as seguintes motivações:

- Efetuar Login
- Adicionar Users;
- Adicionar Clients e Admins;
- Adicionar Credit à conta do Client;
- Adicionar Purchases;
- Adicionar Games, e Copies disponíveis para Compra;
- Adicionar Genres
- Adicionar Platforms
- Adicionar Discounts
- Adicionar Followers e Follows
- Adicionar Companies
- Adicionar Franchises
- Adicionar Reviews ou fazer Update a Reviews já feita
- Atualizar Dados do User
- Atualizar Dados das Franchise,
- Atualizar Dados dos Genres
- Atualizar Dados das Companies
- Atualizar Dados das Platforms
- Filtrar Histórico de Purchases
- Filtrar Histórico de Credits
- Filtrar Dados do Client
- Filtrar Games disponíveis para Compra assim como os seus Franchises, Companies, Genres, Platforms ou se o Jogo se encontrava em disconto ou Não.
- Remover Follows e Followers.
- Remover Discounts e Genres associados a um Game

Na imagem seguinte, demonstra-se um Exemplo de Utilização de Stored Procedures, a SP que insere uma nova Compra Efetuada à Tabela Purchase,utilizando uma Transaction(irá ser falada mais tarde) e verificando se existem cópias disponíveis de um Game para a Plataforma desejada na Compra (caso não, é adicionada uma nova). Além disso atualiza o Saldo disponível do utilizador.

```
go
CREATE PROCEDURE Project.pd_insertPurchase(
    @PurchaseDate DATE,
    @IDClient INT,
    @IDGame INT,
    @PlatformName VARCHAR(30),
    @res VARCHAR(35) output
    )
    AS
        BEGIN
            BEGIN TRAN
            DECLARE @Price DECIMAL(5,2)
            DECLARE @countDispCopies INT;
            DECLARE @SerialNum INT;
            DECLARE @tempPer INT;
            BEGIN TRY
            IF ( (SELECT Project.udf_checkUserPurchase(@IDClient,@IDGame)) = 1)
                raiserror ('User Already Contains that Game',16,1);

            SET @Price = (SELECT Price from Project.Game WHERE Game.IDGame=@IDGame)
            SET @SerialNum= (SELECT top 1 notBought FROM Project.[udf_checkGameCopies] (@IDGame,@PlatformName))
            IF @SerialNum IS NULL
                BEGIN
                    INSERT INTO Project.Copy VALUES (@IDGame,@PlatformName)
                    SET @SerialNum =( SELECT TOP 1 Copy.SerialNum FROM Project.[Copy] WHERE Copy.IDGame=@IDGame AND Copy.PlatformName=@PlatformName ORDER BY SerialNum DESC  )
                END

            SET @tempPer = (SELECT TOP 1 * FROM Project.[udf_checkGameDiscount](@IDGame))
                IF @tempPer is not null
                    BEGIN
                        SET @Price-=@Price*(@tempPer/100)
                    END
            INSERT INTO Project.Purchase(Price,PurchaseDate,IDClient,SerialNum) VALUES (@Price,@PurchaseDate,@IDClient,@SerialNum)
            SET @res ='Success!'
            END TRY
            BEGIN CATCH
                 SET @res= ERROR_MESSAGE()
                 rollback tran
            END CATCH
            if @@TRANCOUNT >0
            COMMIT TRAN
        END
```

Decidimos utilizar SP's para filtrar Dados, consoante as opções do utilizador, devido a grande eficiência destes ao reduzir significativamente a carga no sistema e tempos de carregamento dos Dados. Em casos que envolveram a inserção de Dados ou atualização de Dados em várias tabelas numa única procedure procedeu-se a utilização de **Transactions** com vista a que se ocorresse erro toda a operação seria alvo de um *rollback.*

User Defined Functions - UDF

Como já foi referido anteriormente, foi dada a preferência a UDFs em relação às Views (que não foram usadas). Os casos de uso, maioritariamente necessitavam da passagem de argumentos o que contribuiu para o uso destas. Utilizamos UDFs assim para processar a maior parte dos pedidos de leitura dos Dados da Base de Dados, tendo sido utilizadas seguintes os Tipos de UDFs : Escalares, *Inline Table -Valued*.

Com as UDFs, alcançamos uma abstração nas leituras realizadas a Base de Dados. Estas tiveram assim, as seguintes motivações e objetivos:

- Verificar se um Email já se encontrava em uso;
- Verificar se um Username já se encontrava em uso;
- Verificar se um **User** é **Admin**;
- Verificar se um User é **Client**;
- Verificar se um **Client** tem um jogo em comum com outro Client;
- Verificar se um **Client** segue Outro;
- Retornar os Jogos Adquiridos de um Client;
- Retornar os **Followers** de um Client;
- Retornar os Todos os Jogos em comum entre dois Clients;
- Retornar os **Genres** dos Games;
- Retornar as Plataformas de um **Game**
- Retornar os Detalhes de uma **Purchase**;
- Retornar a lista de Reviews de um Game;
- Retornar a lista de Jogos,mais e menos vendidos.
- Retornar **Copies** disponíveis de um Game.
- Retornar um **Discount** Válido de um Game.
- Entre outras (no total cerca de 38 UDFs)

Na imagem seguinte, encontra-se um exemplo de utilização de uma UDF, que retorna a percentagem de Desconto (**Discount**) de um Jogo, caso este seja válido para o Jogo em causa.

```
CREATE FUNCTION Project.[udf_checkGameDiscount] (@IDGame INT) RETURNS TABLE
AS
    RETURN (SELECT [Percentage] AS [Percentage] FROM Project.Game
    JOIN Project.DiscountGame ON Game.IDGame =DiscountGame.IDGame
    JOIN Project.Discount ON Discount.PromoCode =DiscountGame.PromoCode
    WHERE DATEDIFF(DAY,DateEnd,GETDATE()) <0  AND DATEDIFF(DAY,DateBegin,GETDATE()) >0 AND Game.IDGame=@IDGame)

GO
```

Triggers

Decidimos utilizar triggers **instead of insert** na nossa Base de Dados, com o objetivo de garantir que na maior parte das operações de insert eram válidas. Por exemplo, sempre que inserimos **Games**,**Genres**,**Franchises**,etc é verificado no trigger se o nome do Game,Genre,Franchise já se encontram nas suas respetivas Tabelas. Além disso, nos casos de inserir uma nova Compra na Base de Dados, por exemplo, o trigger tem a função de confirmar se o Cliente possui saldo suficiente na Conta para efetuar a transação, se sim, atualiza-se o saldo e confirma-se a Compra, em caso contrário, impede-se a continuação da operação. Por opção não utilizámos triggers nas operações de delete.
Na imagem a seguir, encontra-se aquele que é o Trigger por detrás da inserção de **Credit** na conta de um **Client**.

```
go
CREATE TRIGGER Project.trigger_credit ON Project.Credit
INSTEAD OF INSERT
AS
    BEGIN
            DECLARE @MetCredit as VARCHAR(20);
            DECLARE @DateCredit as DATE;
            DECLARE @ValueCredit as DECIMAL(4,2);
            DECLARE @IDClient AS INT;

            SELECT @MetCredit = MetCredit,@DateCredit = DateCredit, @ValueCredit = ValueCredit,@IDClient = IDClient FROM INSERTED;
                IF NOT EXISTS(select UserID FROM Project.Client WHERE @IDClient=UserID)
                    RAISERROR('User not found!',16,1)
                ELSE
                BEGIN
                    INSERT INTO Project.Credit(MetCredit,DateCredit,ValueCredit,IDClient) VALUES (@MetCredit,@DateCredit,@ValueCredit,@IDClient)
                    UPDATE Project.Client
                        SET Balance+=@ValueCredit
                        WHERE Project.Client.UserID=@IDClient
                END
    END
```

Cursors

Tal como as Views, não utilizamos qualquer tipo de Cursor no desenvolvimento deste Trabalho. O principal motivo, foi sem dúvida o facto de termos conseguir realizar todas as nossas queries através da Álgebra Relacional do SQL Server, pelo que não tentamos forçar o uso destes apenas para colocar neste Trabalho, ou apenas para demonstrar um caso de uso de um Cursor, sem motivo algum. A preferência pela não utilização de Cursores é espelhada nos factos de esta ser mais lenta, além de que dificulta mais algumas coisas, que se fazem facilmente com Álgebra Relacional.

Índices

Por fim, o último tópico que não decidimos abordar no nosso trabalho foram índices, visto que como a Base de Dados que foi construída é relativamente pequena, dificilmente teríamos problemas de eficiência na execução de Queries. Assim a utilização de Índices não foi necessária do nosso ponto de vista. Porém é necessário realçar que se fosse necessário escalar esta base de dados seriam necessários índices de maneira a diminuir o tempo de execução.

Segurança

Como já foi referido neste Documento, como forma de estabelecer uma maior segurança à Base de Dados, utilizou -se SQL Parametrizado, impedindo assim o uso de *queries ad-hoc.* Além disso, na Tabela **User,** decidiu-se guardar as palavras-Passe dos utilizadores recorrendo a função **ENCRYPTBYPASSPHRASE**(), que permite a encriptação das palavras passes de uma forma muito simples, o que apesar de não ser a forma mais segura, fornece alguma segurança. Assim, o tipo de Dados associado a este atributo é **VARBINARY** (Binary Byte String). Na primeira imagem, encontra-se a tabela **User**, e na segunda imagem, encontra-se uma Procedure que regista os Dados de um novo **User** na Plataforma.

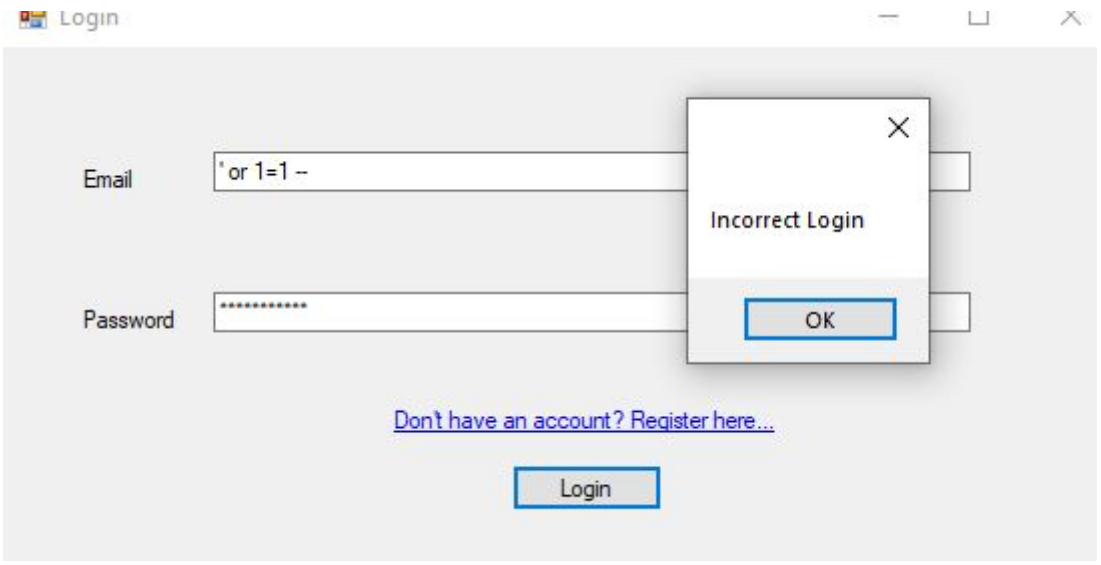
```
CREATE TABLE Project.[User](
    UserID          INT IDENTITY(1,1) ,
    Email           VARCHAR(50) UNIQUE    NOT NULL,
    [Password]        VARBINARY(64)     NOT NULL,
    RegisterDate    DATE            NOT NULL,
    PRIMARY KEY (USERID)
);
```

```
go
create procedure Project.pd_sign_up
    @email VARCHAR(50),
    @password VARCHAR(20),
    @registerDate    DATE ,
    @userName VARCHAR(50),
    @fullName VARCHAR(max),
    @sex        CHAR,
    @birth      DATE,
    @response varchar(255) output
as
begin
    begin try

        insert into Project.[User](Email,[Password], RegisterDate) values (@email,ENCRYPTBYPASSPHRASE('**********',@password) ,@registerDate)
        DECLARE @client_id AS INT;
        SELECT @client_id = UserID from Project.[User] where [User].Email=@email
        INSERT INTO Project.Client(UserID,Username,FullName,Sex,Birth,Balance)  VALUES(@client_id,@userName,@fullName,@sex, @birth,0.0)
        set @response='Success'
    end try
    begin catch
        set @response=ERROR_MESSAGE()
    end catch
end
```

Efetuamos também a validação dos campos de preenchimentos dos formulários na aplicação, antes da execução das SPs.

Por fim, procedeu-se a um conjunto de testes efetuando tentativas de **SQL Injection** na plataforma, com a finalidade de verificar se a Base de Dados se encontrava vulnerável, ou não. Como é possível verificar, nas imagens a seguir, são mostradas algumas mensagens de erro personalizadas, quando se tenta efetuar um ataque deste género, e a utilização de SQL não Dinâmico, respetivamente:




```
String Email = LoginEmailBox.Text;
String Password = LoginPasswordBox.Text;
SqlCommand cm = new SqlCommand("Project.pd_Login");
cm.CommandType = CommandType.StoredProcedure;
cm.Parameters.Add(new SqlParameter("@Loginemail", SqlDbType.VarChar));
cm.Parameters.Add(new SqlParameter("@password", SqlDbType.VarChar));
cm.Parameters.Add(new SqlParameter("@response", SqlDbType.VarChar,1));
cm.Parameters["@LoginEmail"].Value = Email;
cm.Parameters["@password"].Value = Password;
cm.Parameters["@response"].Direction = ParameterDirection.Output;
cm.Connection = Program.cn;
cm.ExecuteNonQuery();
```

Conclusão

Com este trabalho, foi nos possível consolidar os conhecimentos nos tópicos de Base de Dados abordados nesta disciplina assim como desenvolver capacidades na construção de pequenas aplicações que necessitem de Uso de uma Base de Dados.

Além disso, é relevante referir que consideramos que conseguimos alcançar, com sucesso, os objetivos pretendidos com este trabalho, onde o trabalho de equipa foi fundamental para conseguir realizar algumas queries.