myGameStack - Base de Dados de Jogos

Por: Vasco Ramos - nº mec 88931
Diogo Silva - nº mec 89348

Data de Preparação: Aveiro, 02 de Junho de 2019

Cadeira: Base de Dados

Corpo Docente: Professor Carlos Costa
Professor Sérgio Matos

Índice

Introdução

Em seguimento do plano curricular da disciplina Base de Dados, do curso de Engenharia Informática, da Universidade de Aveiro, este relatório é o resultado da execução do trabalho final e tem como objetivo mostrar o que foi feito no mesmo.

As imagens dos diagramas encontram-se disponíveis, no seu formato original, no diretório **Diagrams**.

O script responsável pela instalação da base de dados está disponível no diretório **SQL_setup_script**.

Todo o código relativo à interface encontra-se no diretório **Interface**.

Os scripts de instalação da base de dados estão disponíveis no diretório **SQL_Setup_Script**.

A demo da aplicação está disponível no diretório **Demo**.

Os slides da apresentação do dia 6 estão disponíveis no diretório **Presentation**.

O ficheiro **README.txt** contém a informação necessária para alterar o utilizador da base de dados de forma a permitir o teste do sistema.

Entidades

Entidade	Descrição
User	Utilizador do sistema, é este o principal alvo da plataforma.
Admin	Pessoa responsável por adicionar novos **Games**, **Franchises**, **Genres**, **Developers**, etc. É também responsável pela remoção de utilizadores ou promoção de um dado **User** a **Admin**.
Game	Unidade central da plataforma, o jogo é um dos principais dados de informação do sistema.
Genre	Entidade que representa as categorias a que um jogo pode pertencer (categorias tais como Shooter, Action RPG, entre outras)
Developer	Empresa responsável por desenvolver jogos.
Publisher	Empresa responsável pela publicação de jogos.
Platform	Entidade que representa os ambientes em que os jogos podem são lançados/distribuídos como, por exemplo, Windows 10, PlayStation 4, etc.
Franchise	Entidade que representa os franchises existentes no sistema. Estes franchises agregam um conjunto de jogos sobre uma história base, mundo partilhado, mecânicas de *gameplay* ou, simplesmente, nome comum. Como exemplos temos a série Halo, Resident Evil, Counter-Strike,entre muitas outras.
Tournment	Entidade que representa os torneios existentes. Cada um destes relaciona-se com um dado jogo.
Event	Entidade fraca de **User** que representa as listas que cada jogador tem, onde pode colocar os jogos que quiser.
EventType	Tipo de evento, isto é, tipo de Lista que o utilizador irá manipular. As listas podem ser do tipo *Playing, Dropped, Completed, Plan To Play* ou *WishList*.
Review	Entidade fraca de **Game**, representa o conceito de avaliação que um dado **User** pode dar a um dado **Game**.

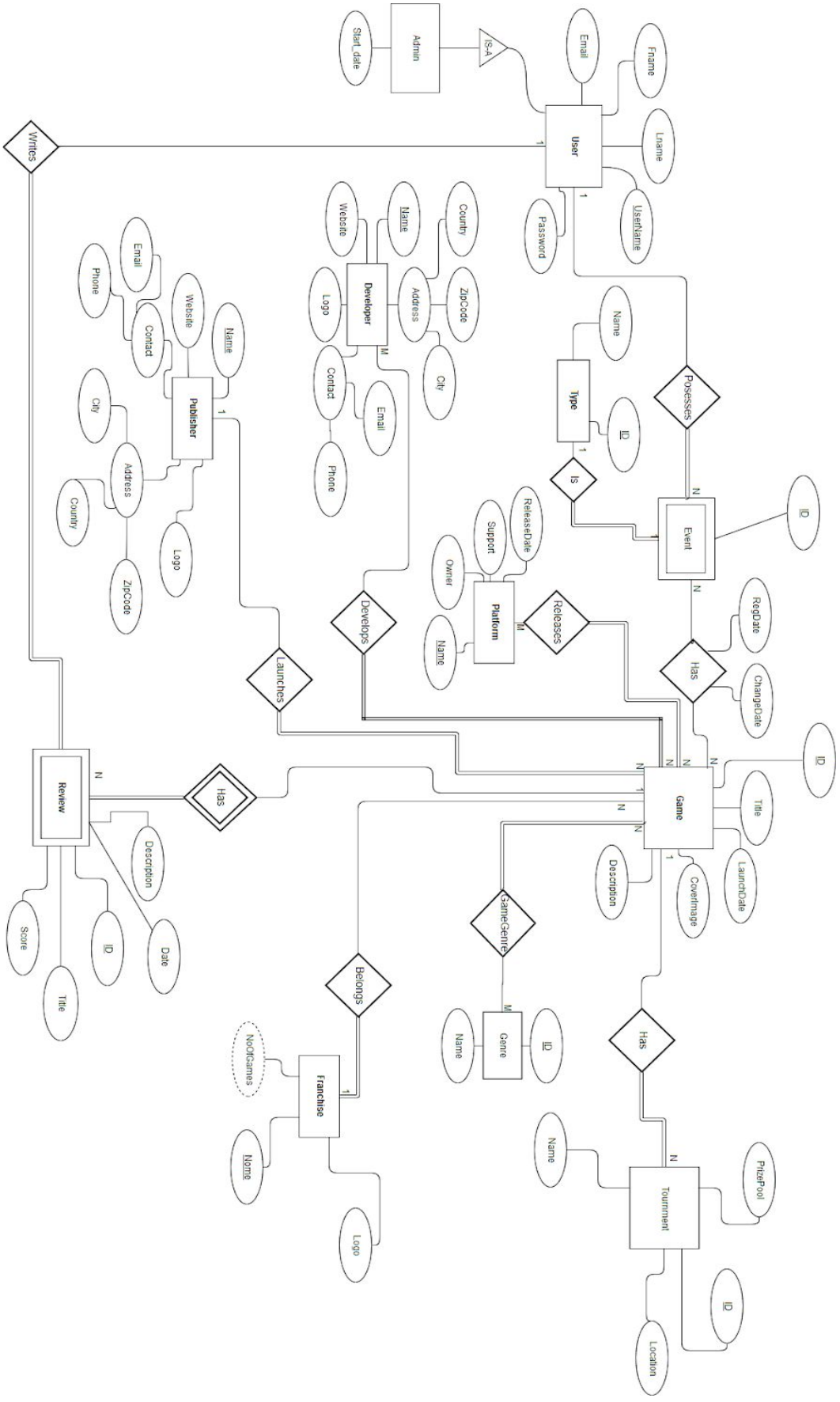
Análise de Requisitos

- Um **User** é identificado pelo um username único, nome (primeiro e último), email e password.
- Um **Admin** é identificado pelo um username único, nome (primeiro e último), email, password e uma data de início de função.
- Um **Game** é identificado por um ID único, título, ano de lançamento, descrição do jogo e imagem de capa.
- Um **Genre** é identificado por um ID único e um nome (também único). Um jogo tem um ou mais genres.
- Um **Developer** é identificado por um ID único, nome (também único), website, contacto (email e telefone), morada (city e country) e logótipo. Um jogo tem um ou mais developers.
- Um **Publisher** é identificado por um ID único, nome (também único), website, contacto (email e telefone), morada (city e country) e logótipo. Um jogo tem um único publisher.
- Uma **Platform** é identificada por um ID único, nome (também único), o seu fabricante/proprietário e a sua data de lançamento no mercado global. Um jogo é lançado em uma ou mais plataformas.
- Um **Franchise** é identificado por um ID único, nome (também único), número de jogos nesse franchise e logótipo. Um jogo pertence a apenas um franchise.
- Um **Tournment** é identificado por um ID único, nome (também único), o seu prize pool, a localização do torneio. Um tournment tem um único jogo. O mesmo jogo pode ter vários tournments associados a si.
- Um **Event** é uma entidade fraca de **User**, identificada por um ID e username do utilizador a quem o evento (lista) pertence.
- Um **EventType** é identificado por um ID único e por um nome (também único). Um Event tem um único EventType.
- Uma **Review** é uma entidade fraca de **Game**, sendo identificada por um ID, um gameID do jogo ao qual a review se associa, username do user que a escreve, uma data, título,

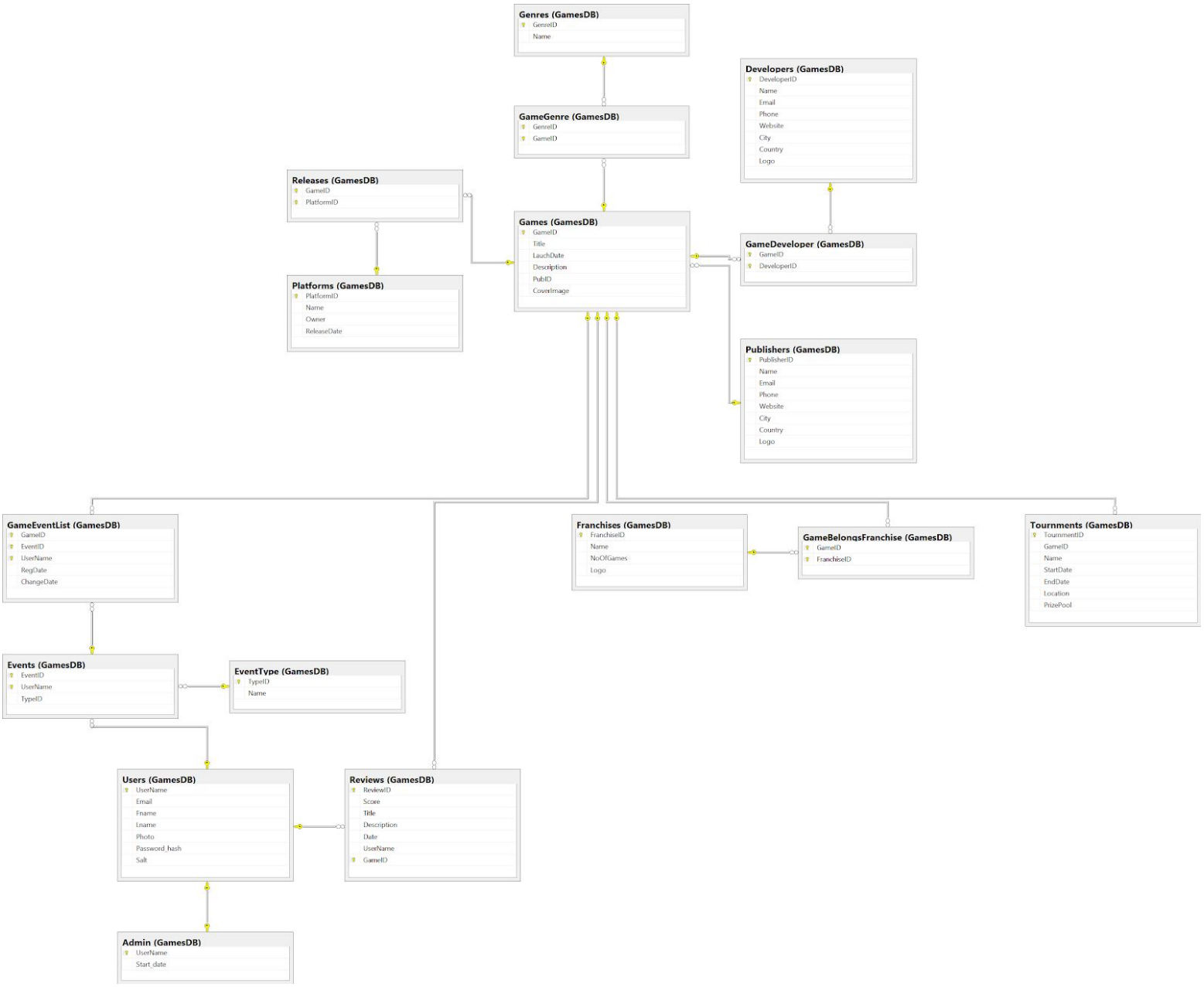
descrição e um score (1 a 5). Um **User** pode escrever várias **Reviews** (apenas uma por cada jogo) , mas só pode escrever uma review nos jogos que estão nas suas várias listas - **Events** - (*Playing, Plan To Play, WishList, Dropped, Completed*).

- Um **User** pode adicionar vários jogos às suas listas - **Events** - como já foi referido no ponto anterior o **User** tem 5 Events: *Playing, Plan To Play, WishList, Dropped, Completed*. Para cada **User** um dado jogo só pode estar numa das 5 listas (***exclusão mútua***).

Diagrama Entidade Relação



Esquema Relacional



Normalização

Tendo em conta que as nossas entidades não tinham muitos atributos, não tivemos entidades não normalizadas, pelo que todas se encontram na 3ª forma normal.

Uma base de dados relacional é constituída por relações entre dados (tabelas). Nestas, tenta-se reduzir o máximo possível a duplicação de dados, bem como a redundância dos mesmos.

Para cumpir com este requesito/necessidade, efetuámos uma série de testes (tal como foi feito nas aulas teóricas e práticas relativas a normalização) com o objetivo de nos certificarmos que a nossa base de dados respeitas as primeira, segunda e terceira formas normais.

Assim, após a realização desses testes, podemos garantir que o nosso modelo de dados se encontra na terceira forma normal.

Data Defining Language (DDL)

Os **Users** necessitam de um login para iniciar uma sessão no sistema, e ,tendo em conta que é má prática guardar informações confidenciais e sensíveis em texto simples, como passwords, guardamos a password do user em *Hash* acrescentando ainda o *Salt* para dificultar ataques do tipo brute-force e ataques de dicionário.

```
create table GamesDB.[Users] (
      UserName        varchar(30)      unique       not null,
      Email           varchar(max)                  not null,
      Fname           varchar(max)                         ,
      Lname           varchar(max)                         ,
      Photo           varchar(max)                         ,
      Password_hash   binary(64)                    not null,
      Salt            uniqueidentifier              not null,

      primary key (UserName)
);
```

Várias entidades como, por exemplo, o User, Game, Developer, entre outros têm uma imagem. A nossa abordagem foi guardar as imagens em *base64* num atributo varchar(max).

```
create table GamesDB.Games (
    GameID          int             identity(1,1)   not null,
    Title           varchar(100)    unique          not null,
    LauchDate       date                                   ,
    [Description]   varchar(max)                           ,
    PubID           int                             not null,
    CoverImage      varchar(max)                           ,

    primary key (GameID)
);
```

Data Manipulation Language (DML)

No nosso sistema usamos SQL parametrizado e também permitimos pesquisa dinâmica sujeita a alguma validação na tentativa de, até certo ponto, evitar *SQL Injection*. Em grande parte do sistema os resultados de pesquisa são apresentados com paginação de forma a diminuir a carga computacional envolvida no loading de informação (diminuindo portanto os tempos de carregamento).

Triggers

Para a base de dados decidimos definir alguns **triggers instead of**, sendo as suas funções principais garantir a integridade dos dados. Estes foram utilizados em ação de *on insert* para situações como a criação de um novo jogo, franchise, developer, etc em que tinha de ser garantido que não existiam dois jogos ou dois franchises (por exemplo) com o mesmo nome.

Outros dois triggers que foram criados são *after insert*: um deles na tabela dos **Users** para criar, para cada user, 5 entradas na tabela **Events** (as 5 listas pessoais que cada user tem) e outro trigger na ligação de um jogo com um franchise para incrementar o atributo *NoOfGames* do franchise de cada vez que é adicionado um jogo a esse.

Ainda de referir que numa primeira abordagem pensámos em criar triggers *instead of delete* para fazer a total remoção da informação relacionada e/ou dependente da entrada de uma dada tabela (por exemplo um jogo) que estava a ser removida, no entanto, acabámos por perceber que a forma mais eficiente de fazer isto era tornar as *constraints* de *foreign keys* tomar o comportamento do ***delete on cascade*** e, portanto, acabámos por optar esta última forma de lidar com a situação.

Stored Procedures

Os **Stored Procedures** na nossa solução foram, maioritariamente, utilizados como uma forma de criar um nível de abstração no acesso à escrita na base de dados.

Sendo que permitimos que sejam realizadas as seguintes operações:

- Adicionar Users,
- Adicionar Admins,
- Adicionar Franchises,
- Adicionar Developers,
- Adicionar Publishers,
- Adicionar Genres,
- Adicionar Tournments,
- Adicionar Platforms,
- Adicionar Reviews,
- Adicionar Games,
- Remover Users,
- Remover Franchises,
- Remover Developers,
- Remover Publishers,
- Remover Genres,
- Remover Tournments,
- Remover Platforms,
- Remover Games.

De toda esta enumeração, é de salientar que as duas SPs que consideramos fulcrais para o nosso trabalho são:

- Adicionar Users: é esta SP que nos permite encriptar a password com uma Hash e o Salt como já foi referido e garantir a segurança destes dados.
- Adicionar Games: esta SP é a mais longa que temos, pois, é esta procedure que garante (com a utilização de uma transaction que será falada mais em detalhe a seguir) que quando adicionamos um jogo, adicionamos também todas as relações desse jogo para garantir coesão na base de dados.

Para além das que já apresentámos, criámos também **Stored Procedures** que permitissem ver as pesquisas de forma paginada:

- Search Franchises,
- Search Developers,
- Search Publishers,
- Search Tournments,
- Search Platforms,
- Filter Franchises,
- Filter Developers,
- Filter Publishers,
- Filter Tournments,
- Filter Platforms.

Para esta leitura de dados decidimos usar SPs, pois, estas permitem mais operações e permitem retornar dados diretamente de selects o que foi uma vantagem na escrita da query. À semelhança das anteriores, estas SPs são também importantíssimas, porque reduzem imenso a carga do sistema e permitem tempos de *loading* de dados muito mais baixos, o que é uma vantagem para o utilizador.

Por fim, temos também uma **Stored Procedure** para realizar o login de um **User**.

User Defined Functions

As nossas User Defined Functions (**UDF**s) foram usadas para encapsular a Base de Dados, por isso grande parte dos pedidos de leitura da base de dados são realizados usando **UDF**s. Para isto, usámos três tipos de **UDF**s:

- UDFs Escalares,
- Inline Table-valued UDFs
- Multi-Statement table-valued functions.

Desta forma, fomos capazes de esconder toda a lógica por detrás das pesquisas realizadas, facilitando a utilização das mesmas (pesquisas) na construção da interface gráfica do sistema.

A enumeração seguinte contém as **UDF**s que definimos:

- Verificar se um User é Admin.
- Verificar se um jogo faz parte de uma dada lista de um dado User.
- Verificar se um jogo faz parte de alguma das listas de um dado User.
- Obter toda a informação de um dado User (profile).
- Obter os Developers de um dado jogo.
- Obter as Reviews de um dado jogo.
- Obter as Plataformas de um dado jogo.
- Obter a Review escrita por um dado User num dado Jogo.
- Obter a informação detalhada de um Jogo + a informação detalhada do seu Publisher.
- Obter uma average score de um dado jogo.
- Obter os Genres de um dado Jogo.

As UDFs que mais se destacam acabam por ser as três primeiras que, apesar de não serem as mais complexas, são essenciais para o correto funcionamento do sistema, pois, são queries de validação de contextos que nos permitem garantir que as restrições do modelo de dados estão a ser respeitadas e não estamos a criar dados inconsistentes nem permitir acessos indevidos a certas funcionalidades do sistema.

Índices

Na atribuição dos **índices** tivemos em atenção a frequência das *queries* e quais destas é que iriam beneficiar de um (ou mais) índices. Após esta análise, concluímos que precisávamos de apenas mais um índice: **Índice NonClustered (GameID) de Tournment**. Isto deve-se ao facto de este ser uma de duas formas de filtrar tournments (através do jogo associado ao **Tournment**).

O facto de só termos considerado necessário um índice tem a ver com a facto de, para garantirmos que cada entidade tivesse um nome único (não pode haver 2 jogos com o mesmo nome, ou 2 tournments com o mesmo nome - aplica-se a mais entidades) decidimos por o *Name* como atributo *unique* e *not null*; por isto o SGBD cria, automaticamente, um índice para cada um destes *uniques*. Ora, tendo em conta que a principal forma de pesquisa é pelo nome, a necessidade de criar índices para estes atributos já tinha sido suprida pelo fenómeno explicado anteriormente.

Notas Finais

Antes de tecer as últimas conclusões, é de salientar algumas opções que foram tomadas durante a execução deste trabalho:

- Não fizemos Views: não só por todas as situações de pesquisa se relacionarem com opções do utilizador, isto é, passagem de parâmetros, pelo que a utilização de uma **UDF** era mais adequada, mas também pelo facto de, como vimos nas aulas, as **UDF**s terem uma melhor performance que as **Views**.
- Não fizemos Cursores: no desenvolvimento do projeto não nos deparámos com nenhuma situação que a álgebra relacional não conseguisse resolver, pelo que optámos sempre pela álgebra relacional (que é bastante mais eficiente), ao invés de forçar, para um dado contexto de acesso aos dados, o uso de cursores com apenas o intuito de mostrar que o sabíamos fazer (não significando que não soubéssemos) apesar de ser perfeitamente possível suprir a necessidade desse contexto com AR.

Relativamente a tudo o que foi feito, gostávamos de salientar as duas *queries* que se revelaram os maiores desafios:

- A inserção de um jogo: como um jogo tem tantas relações (developers, publisher, genres, platforms, Tournments, etc etc), garantir que tudo corria bem durante a inserção de toda a informação (e o ***rolllback*** de toda a operação caso corresse mal) acabou por ser uma das maiores **Stored Procedures** que foi feita no trabalho (neste houve claramente a utilização de uma **transaction**)
- A inserção de um jogo numa dada lista ou mover um jogo de uma lista do User para outra das suas listas: devido a todas as verificações que necessariamente têm de ser feitas.

Conclusão

A realização do trabalho em discussão foi de elevada importância para o desenvolvimento dos conhecimentos e capacidades que devem ser adquiridos na cadeira de Base de Dados, pois, apesar da execução dos exercícios das aulas práticas, a realização de um trabalho deste tipo que acaba por ter uma dimensão maior do que a dimensão dos exercícios das aulas permite-nos praticar e compreender mais profundamente os conceitos lecionados e, o mais importante de tudo, permite-nos aprender através dos nossos erros e perceber quais são as boas práticas no desenvolvimento de uma base de dados.

Neste trabalho, o nosso principal objetivo era criarmos um sistema que fosse capaz de suportar todas as funcionalidades de um sistema de livraria de jogos (uma espécie de IMDb para jogos), pelo que não nos focámos no desenho uma interface gráfica muita apelativa (até porque acaba por estar fora do âmbito da disciplina), mas focámo-nos na conceção e implementação de uma base de dados robusta com a abstração de dados suficiente para ser possível aceder aos dados sem atacar diretamente a mesma

Olhando em retrospectiva, após a conclusão deste trabalho, cremos que todos os objetivos e desafios propostos foram alcançados e que conseguimos criar uma base de dados que vai de encontro às nossas expetativas e à nossa ambição inicial.

Bibliografia

- Enunciado, código e comentários fornecidos pelo Docente.
- Slides Teóricos da cadeira de Base de Dados.