U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Projeto de Bases de Dados (BDAD) - Entrega III

Turma 1 – Grupo 106

"Plataforma de *Streaming* de Filmes e Séries - ***Netflix&Chill***"

Mestrado Integrado em Engenharia Informática e Computação – 2ºAno 2ºSemestre

(2020/2021)

Fevereiro 2021 – Maio 2021

Ana Matilde Guedes Perez da Silva Barra up201904795
Maria Beatriz Russo Lopes dos Santos up201906888
Tiago Caldas da Silva up201906045

Índice

Introdução ao tema

No âmbito da unidade curricular *Base de Dados* do curso *Mestrado Integrado em Engenharia Informática e Computação* (**MIEIC**), optamos por uma exequível e adaptada representação da plataforma de *streaming Netflix*, como tema para o projeto.

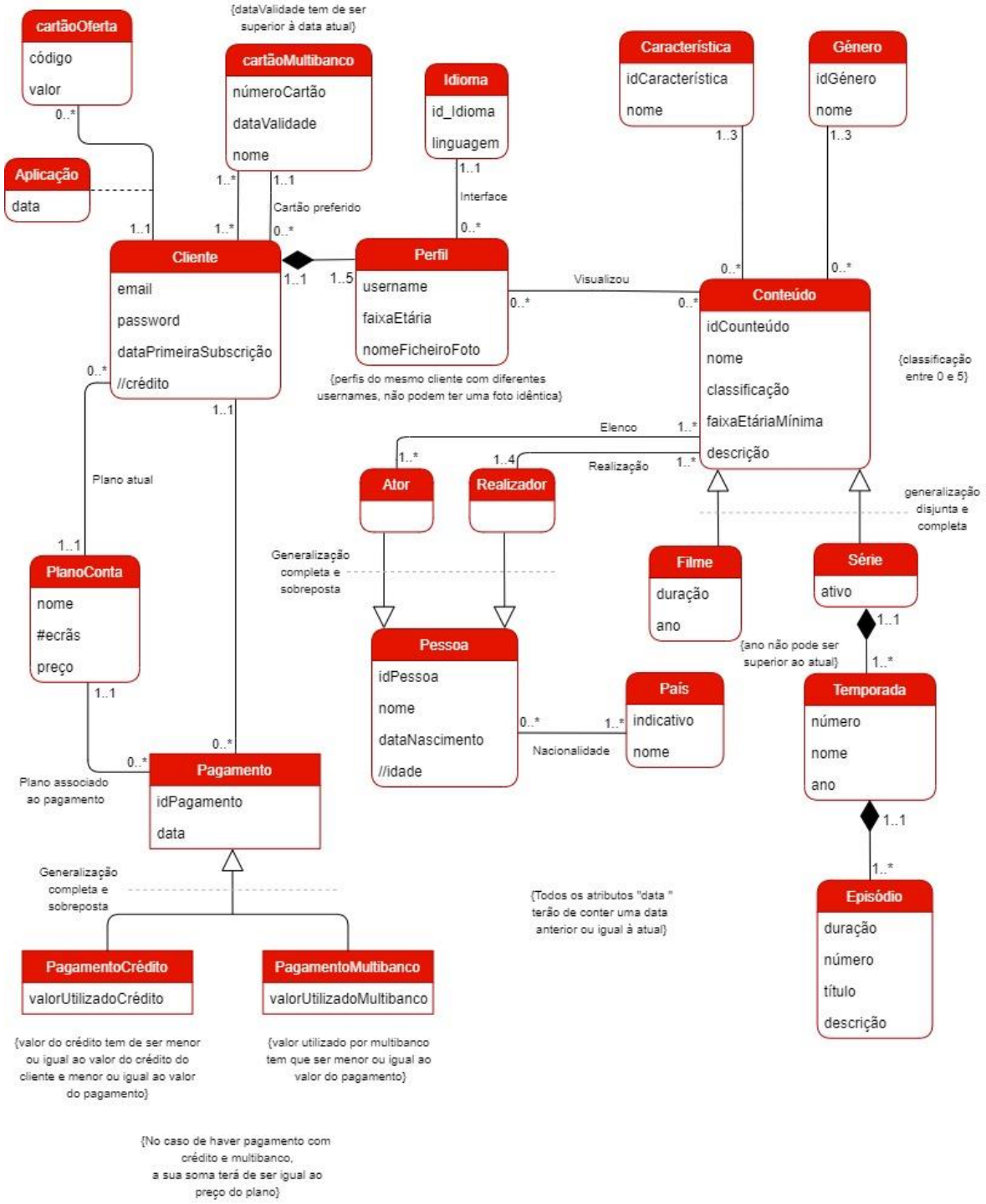
1. Numa **primeira fase**, começamos por modelar conceptualmente aquele que vai ser a nossa base de dados:
    - Armazenamos as informações referentes aos **clientes** e seus **perfis**, possíveis **conteúdos** (**filmes** e **séries**) disponíveis na plataforma, tal como **atores** e **realizadores** intervenientes nos mesmos.

2. Na **segunda parte** do projeto, efetuamos algumas alterações no Modelo Conceptual, acrescentando, por exemplo:
    - A classe **Imagem**, com ligação a **Conteúdo**, **Episodio** e **Perfil**;
    - Ligação entre **Perfil**-**Episodio** e **Perfil**-**Filme** de forma a identificar se um determinado perfil os visualizou;
    - Alteração do atributo *indicativo* em **Pais** para *code_ISO*, uma vez que verificamos a existência de países com o mesmo indicativo (Ex: Estados Unidos e Canadá);
    - Remoção da ligação entre **PlanoConta**-**Pagamento**.
    - Remoção do atributo *valor* em **Pagamento**.
    - Adição de algumas restrições.
    - Os pagamentos mensais de um **Cliente** serão iniciados no momento da sua adesão.

3. Na **terceira e última parte** do projeto foram definidas um conjunto de 10 interrogações pertinentes para o contexto da base de dados. Para além disso, de forma a manter uma boa monitorização e manutenção da base de dados, foram implementados 3 gatilhos. Por fim, conforme o *feedback* da segunda entrega, alteramos o diagrama UML.

Ana Matilde Guedes Perez da Silva Barra up201904795
Maria Beatriz Russo Lopes dos Santos up201906888
Tiago Caldas da Silva up201906045

Diagrama UML (Entrega I)



Ana Matilde Guedes Perez da Silva Barra up201904795
Maria Beatriz Russo Lopes dos Santos up201906888
Tiago Caldas da Silva up201906045

Descrição do diagrama (Entrega I)

Cliente

Esta classe armazena os dados referentes ao cliente da plataforma, tal como o **Crédito**, por defeito com valor igual a 0.00€, que aumenta a cada aplicação de um cartão de oferta. Caso o cliente possua crédito no momento do pagamento, este é utilizado totalmente (se menor ou igual à mensalidade) ou parcialmente (se superior).

CartãoOferta

Classe que influencia o crédito do cliente. Existe a possibilidade de aplicar múltiplos cartões. A plataforma possui armazenamento dos que não foram aplicados, sendo que os que foram têm uma data de ativação associada.

CartãoMultibanco

Não tem qualquer influência no crédito do cliente. Existe a possibilidade de um cliente ter associado um ou mais cartões, porém, apenas um poderá ser definido como preferido (aquele que é utilizado nos pagamentos).

PlanoConta

Cada cliente tem associado um plano de *streaming* de três disponíveis:

- Plano Base: permite a visualização concomitante de um ecrã, com uma quota mensal de 7.99€;
- Plano Standard: permite a visualização concomitante de dois ecrãs, com uma quota mensal de 10.99€;
- Plano Premium: permite a visualização concomitante de quatro ecrãs, com uma quota mensal de 13.99€.

A classe encontra-se associada a **Pagamento** de forma a associar um valor ao pagamento. Para além disso, torna possível uma gestão do histórico de planos do cliente, uma vez que este pode optar por o alterar.

Pagamento

Esta classe é a superclasse de **PagamentoCrédito** e **PagamentoMultibanco** com o objetivo de definir a forma de pagamento.

Na existência de crédito, este é sempre utilizado obrigatoriamente em primeiro lugar. Deste modo, caso o valor não salde a dívida, o montante restante é retirado do cartão multibanco definido como preferido.

O primeiro pagamento é feito um mês após a adesão do cliente, sendo os seguintes efetuados após 30 dias do seu anterior.

Perfil

Cada perfil é ainda caracterizado por uma **faixa etária**, que limita o conteúdo disponível para visualização. Perfis do mesmo cliente com diferentes *username* não podem ter uma foto idêntica.

Ana Matilde Guedes Perez da Silva Barra up201904795
Maria Beatriz Russo Lopes dos Santos up201906888
Tiago Caldas da Silva up201906045

Conteúdo

Esta classe é uma generalização dos **filmes** e **séries** disponíveis na plataforma, englobando todos os atributos comuns a ambos, de entre os quais a **FaixaEtáriaMinima** que indica qual a menor faixa etária permitida para visualizar o conteúdo.

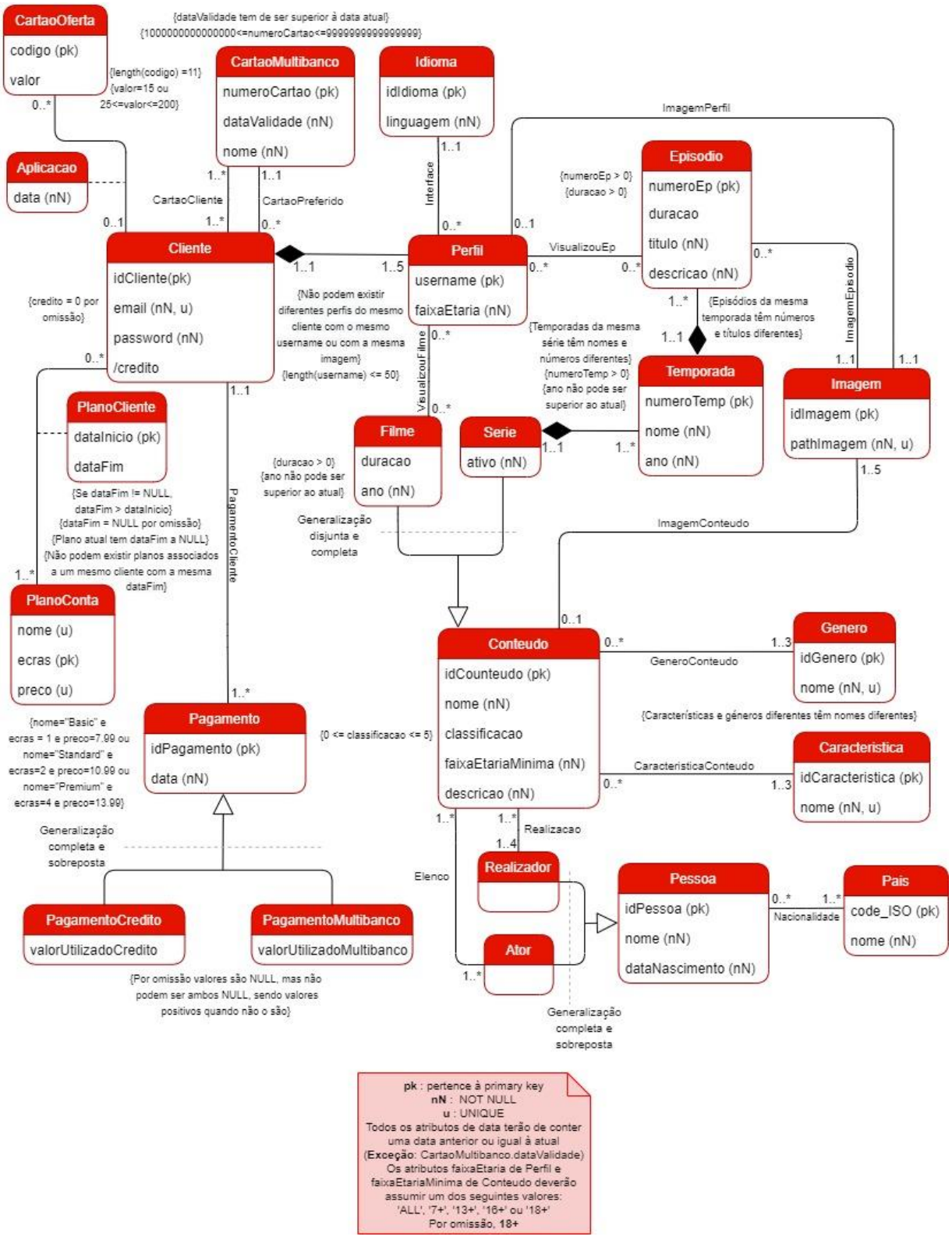
Característica

Esta classe guarda um possível adjetivo para caracterizar um conteúdo.
Exemplo: emocionante, espirituoso, bizarro...

Série

O atributo **ativo** indica se a série se encontra ativa, ou seja, se continua em produção.

Diagrama UML (Entrega II)



Esquema Relacional (Entrega II)

Sublinhado: simboliza que o atributo pertence à **chave primária.**

Atributo → Classe: simboliza que o atributo é uma **chave estrangeira.**

Cliente (idCliente, email, password, credito)

PlanoConta (ecras, nome, preco)

PlanoCliente (dataInicio, dataFim, idCliente → Cliente, ecras → PlanoConta)

Pagamento (idPagamento, data, valorUtilizadoCredito, valorUtilizadoMultibanco) **[1]**

PagamentoCliente (idCliente → Cliente, idPagamento → Pagamento)

CartaoOferta (codigo, valor)

Aplicacao (codigo → CartaoOferta, idCliente → Cliente, data)

CartaoMultibanco (numeroCartao, dataValidade, nome)

CartaoCliente (idCliente → Cliente, numeroCartao → CartaoMultibanco)

CartaoPreferido (idCliente → Cliente, numeroCartao → CartaoMultibanco)

Perfil (idCliente → Cliente, username, faixaEtaria, idImagem → Imagem)

Idioma (idIdioma, linguagem)

Interface ([username, idCliente] → Perfil, idIdioma → Idioma)

Caracteristica (idCaracteristica, nome)

CaracteristicaConteudo (idCaracteristica → Caracteristica, idConteudo → Conteudo)

Genero (idGenero, nome)

GeneroConteudo (idGenero → Genero, idConteudo → Conteudo)

Imagem (idImagem, pathImagem)

ImagemConteudo (idImagem → Imagem, idConteudo → Conteudo)

ImagemEpisodio ([idConteudo, numeroTemp, numeroEp] → Episodio, idImagem → Imagem)

Ana Matilde Guedes Perez da Silva Barra up201904795
Maria Beatriz Russo Lopes dos Santos up201906888
Tiago Caldas da Silva up201906045

Conteudo (idConteudo, nome, classificacao, faixaEtariaMinima, descricao) **[2]**

Pessoa (idPessoa, nome, dataNascimento) **[3]**

Ator (idPessoa → Pessoa) **[3]**

Elenco (idPessoa → PessoaAtor, idConteudo → Conteudo)

Realizador (idPessoa → Pessoa) **[3]**

Realizacao (idPessoa → PessoaRealizador, idConteudo → Conteudo)

Pais (code_ISO, nome)

Nacionalidade (code_ISO → Pais, idPessoa → Pessoa)

Filme (idConteudo → Conteudo, duracao, ano) **[2]**

Serie (idConteudo → Conteudo, ativo) **[2]**

Temporada (idConteudo → Serie, numeroTemp, nome, ano)

Episodio ([idConteudo, numeroTemp] → Temporada, numeroEp, duracao, titulo, descricao)

VisualizouEp ([username, idCliente] → Perfil, [idConteudo, numeroTemp, numeroEp] → Episodio)

VisualizouFilme ([username, idCliente] → Perfil, idConteudo→ Filme)

Algumas estratégias de conversão utilizadas (Entrega II)

[1] A generalização entre **Pagamento**, **PagamentoMultibanco** e **PagamentoCredito** é classificada como uma generalização completa e sobreposta. Deste modo, dado que para o mesmo pagamento, um cliente pode efetuá-lo através do cartão multibanco ou do valor de crédito (havendo a possibilidade de serem ambos aplicados em simultâneo (frequente)), optamos por aplicar a técnica de conversão, uso de *nulls*. Desta forma, apenas existe uma tabela para o pagamento e como é uma generalização completa, no máximo, apenas um dos valores (**valorUtilizadoCredito** ou **valorUtilizadoMultibanco**), é nulo.

[2] A generalização entre **Conteudo**, **Filme** e **Serie** é classificada como uma generalização completa e disjunta. Assim, como todo o **Conteudo** ou é **Filme** ou é **Serie**, optamos pela utilização da estratégia de conversão *E/R style*. Apesar de, inicialmente, termos considerado em utilizar a técnica *Object-Oriented*, como a classe **Conteudo** possui várias associações com outras classes, torna-se mais viável possuir uma *foreign key* para **Conteudo** nas tabelas **Filme** e **Serie** e em qualquer outra tabela correspondente a uma associação entre **Conteudo** e outra classe.

[3] A generalização entre **Pessoa**, **Ator** e **Realizador** é uma generalização completa e sobreposta. Visto que uma **Pessoa**, na nossa base de dados, apenas pode ser um **Ator,** um **Realizador**, ou ambos (não muito comum), optamos pela utilização da estratégia de conversão *E/R style*. Como não é muito usual uma **Pessoa** ser um **Ator** e **Realizador**, no caso de termos optado pela estratégia do uso de *nulls*, grande parte das instâncias da tabela iriam ter um dos atributos (exemplo: éAtor ou éRealizador) não definido. Assim, as tabelas **Ator** e **Realizador** apenas terão uma *foreign key* para **Pessoa**. Na eventualidade de uma **Pessoa** possuir as duas profissões, ambas as tabelas possuirão uma *foreign key* para a mesma **Pessoa.**

Relações: Formas normais (Entrega II)

Para o estudo das dependências funcionais vamos ter de considerar as seguintes propriedades que visam, principalmente, a organização da base de dados para reduzir a redundância dos dados, aumentar a sua integridade e o desempenho. Assim:

1ª Forma Normal (*1NF*):

Nesta forma os atributos precisam de ser atómicos, não devendo existir tabelas que possuam nem valores repetidos, nem atributos possuindo mais do que um valor nesse domínio.

2ª Forma Normal (*2NF*):

Para estar na ***2NF***, também é necessário cumprir os requisitos da ***1NF***. Esta forma define que os atributos não primos (atributos que não pertencem a nenhuma chave) apenas devem depender de um subconjunto da chave candidata.

3ª Forma Normal (*3NF*):

Tal como na forma anterior a esta, para estar na ***3NF***, também é necessário cumprir os requisitos da ***2NF*** (e consequentemente da ***1NF***). Para além disso, para estar na ***3NF*** é necessário cumprir pelo menos uma das seguintes propriedades:

- Todos os atributos não primos são funcionalmente dependentes de todas as chaves candidatas de uma forma não transitiva;
- Uma relação ***R*** está na ***3NF*** se, para cada dependência $\bar{A} \rightarrow \bar{E}$, não trivial, $\bar{A}$ é uma *super-key* **ou** $\bar{E}$ apenas consiste em atributos primos (atributos que pertencem a alguma chave).

Forma Normal de *Boyce-Codd* (*BCNF*):

Uma relação ***R***, encontra-se na ***BCNF***, se, para toda a dependência funcional $\bar{A} \rightarrow \bar{E}$, não trivial, $\bar{A}$ é uma *super-key*, ou seja, $\bar{A}^{+}$ engloba todos os atributos de ***R***.

É importante considerar que todas as relações que respeitem as regras impostas pela ***BCNF***, também respeitam as imposições da ***3NF*** (e consecutivamente da ***2NF*** e ***1NF***).

Assim, contendo este prévio conhecimento, na análise de todas as relações e suas dependências funcionais, vamos identificar aquelas que se encontram na ***BCNF*** e/ou na ***3NF*** identificando possíveis violações naquelas que possam não cumprir as propriedades já referidas.

Ana Matilde Guedes Perez da Silva Barra up201904795
Maria Beatriz Russo Lopes dos Santos up201906888
Tiago Caldas da Silva up201906045

Dependências funcionais (Entrega II)

Cliente (idCliente, email, password, credito)

Possíveis chaves: {idCliente} ou {email}

idCliente → email, password, credito
email → idCliente, password, credito

BCNF	✔
3NF	✔

Em ambas as dependências funcionais o lado esquerdo é uma *super-key*, logo esta relação encontra-se na ***BCNF*** e consecutivamente na ***3NF***.

PlanoConta (ecras, nome, preco)

Possíveis chaves: {ecras}, {nome} ou {preco}

ecras → nome, preco
nome → ecras, preco
preco → nome, ecras

BCNF	✔
3NF	✔

Em todas as dependências funcionais o lado esquerdo é uma *super-key*, logo esta relação encontra-se na ***BCNF*** e consecutivamente na ***3NF***.

PlanoCliente (dataInicio, dataFim, idCliente → Cliente, ecras → PlanoConta)

Possíveis chaves: {idCliente, dataInicio} ou {idCliente, dataFim}

idCliente, dataInicio → dataFim, ecras
idCliente, dataFim → ecras, dataInicio

BCNF	✔
3NF	✔

Em ambas as dependências funcionais o lado esquerdo é uma *super-key*, logo esta relação encontra-se na ***BCNF*** e consecutivamente na ***3NF***.

Ana Matilde Guedes Perez da Silva Barra up201904795
Maria Beatriz Russo Lopes dos Santos up201906888
Tiago Caldas da Silva up201906045

Pagamento (idPagamento, data, valorUtilizadoCredito, valorUtilizadoMultibanco)

Possíveis chaves: {idPagamento}

idPagamento → data, valorUtilizadoCredito, valorUtilizadoMultibanco

BCNF	✔
3NF	✔

O lado esquerdo da dependência funcional é uma *super-key*, logo esta relação encontra-se na **BCNF** e, consecutivamente, na **3NF**.

PagamentoCliente (idCliente → Cliente, idPagamento → Pagamento)

Possíveis chaves: {idPagamento}

idPagamento → idCliente

BCNF	✔
3NF	✔

O lado esquerdo da dependência funcional é uma *super-key*, logo esta relação encontra-se na **BCNF** e, consecutivamente, na **3NF**.

CartaoOferta (codigo, valor)

Possíveis chaves: {codigo}

codigo → valor

BCNF	✔
3NF	✔

O lado esquerdo da dependência funcional é uma *super-key*, logo esta relação encontra-se na **BCNF** e, consecutivamente, na **3NF**.

Aplicacao (codigo → CartaoOferta, idCliente → Cliente, data)

Possíveis chaves: {codigo}

codigo → idCliente, data

BCNF	✔
3NF	✔

O lado esquerdo da dependência funcional é uma *super-key*, logo esta relação encontra-se na **BCNF** e, consecutivamente, na **3NF**.

CartaoMultibanco (numeroCartao, dataValidade, nome)

Possíveis chaves: {numeroCartao}

numeroCartao → dataValidade, nome

BCNF	✔
3NF	✔

O lado esquerdo da dependência funcional é uma *super-key*, logo esta relação encontra-se na **BCNF** e, consecutivamente, na **3NF**.

CartaoCliente (idCliente → Cliente, numeroCartao → CartaoMultibanco)

Possíveis chaves: {idCliente, numeroCartao}

BCNF	✔
3NF	✔

Sem dependências funcionais não triviais, logo não apresenta qualquer violação à **BCNF** e à **3NF**.

CartaoPreferido (idCliente → Cliente, numeroCartao → CartaoMultibanco)

Possíveis chaves: {idCliente}

idCliente → numeroCartao

BCNF	✔
3NF	✔

O lado esquerdo da dependência funcional é uma *super-key*, logo esta relação encontra-se na **BCNF** e, consecutivamente, na **3NF**.

Perfil (idCliente → Cliente, username, faixaEtaria**,** idImagem → Imagem)

Possíveis chaves: {idCliente, username} ou {idCliente, idImagem}

idCliente, username → faixaEtaria, idImagem
idCliente, idImagem → username, faixaEtaria

BCNF	✔
3NF	✔

Em ambas as dependências funcionais o lado esquerdo é uma *super-key*, logo esta relação encontra-se na ***BCNF*** e consecutivamente na ***3NF***.

Idioma (idIdioma, linguagem)

Possíveis chaves: {idIdioma}, {linguagem}

idIdioma → linguagem

linguagem → idIdioma

BCNF	✔
3NF	✔

O lado esquerdo da dependência funcional é uma *super-key*, logo esta relação encontra-se na **BCNF** e, consecutivamente, na **3NF**.

Interface ([username, idCliente] → Perfil, idIdioma → Idioma)

Possíveis chaves: {username, idCliente}

username, idCliente → idIdioma

BCNF	✔
3NF	✔

O lado esquerdo da dependência funcional é uma *super-key*, logo esta relação encontra-se na **BCNF** e, consecutivamente, na **3NF**.

Caracteristica (idCaracteristica, nome)

Possíveis chaves: {idCaracteristica}, {nome}

idCaracteristica → nome
nome → idCaracteristica

BCNF	✔
3NF	✔

Em ambas as dependências funcionais o lado esquerdo é uma *super-key*, logo esta relação encontra-se na ***BCNF*** e consecutivamente na ***3NF***.

CaracteristicaConteudo (idCaracteristica → Caracteristica, idConteudo → Conteudo)

Possíveis chaves: {idCaracteristica, idConteudo}

BCNF	✔
3NF	✔

Sem dependências funcionais não triviais, logo não apresenta qualquer violação à **BCNF** e à **3NF**.

Genero (idGenero, nome)

Possíveis chaves: {idGenero}, {nome}

idGenero → nome
nome → idGenero

BCNF	✔
3NF	✔

Em ambas as dependências funcionais o lado esquerdo é uma *super-key*, logo esta relação encontra-se na ***BCNF*** e consecutivamente na ***3NF***.

GeneroConteudo (idGenero → Genero, idConteudo → Conteudo)

Possíveis chaves: {idGenero, idConteudo}

BCNF	✔
3NF	✔

Sem dependências funcionais não triviais, logo não apresenta qualquer violação à **BCNF** e à **3NF**.

Imagem (idImagem, pathImagem)

Possíveis chaves: {idImagem}, {pathImagem}

idImagem → pathImagem
pathImagem → idImagem

BCNF	✔
3NF	✔

Em ambas as dependências funcionais o lado esquerdo é uma *super-key*, logo esta relação encontra-se na ***BCNF*** e consecutivamente na ***3NF***.

ImagemConteudo (idImagem → Imagem, idConteudo → Conteudo)

Possíveis chaves: {idImagem}

idImagem → idConteudo

BCNF	✔
3NF	✔

O lado esquerdo da dependência funcional é uma *super-key*, logo esta relação encontra-se na **BCNF** e, consecutivamente, na **3NF**.

ImagemEpisodio ([idConteudo, numeroTemp, numeroEp] → Episodio, idImagem → Imagem)

Possiveis chaves: {idConteudo, numeroTemp, numeroEp}

idConteudo, numeroTemp, numeroEp → idImagem

BCNF	✔
3NF	✔

O lado esquerdo da dependência funcional é uma *super-key*, logo esta relação encontra-se na **BCNF** e, consecutivamente, na **3NF**.

Conteudo (idConteudo, nome, classificacao, faixaEtariaMinima, descricao)

Possíveis chaves: {idConteudo}

idConteudo → nome, classificacao, faixaEtariaMinima, descricao

BCNF	✔
3NF	✔

O lado esquerdo da dependência funcional é uma *super-key*, logo esta relação encontra-se na **BCNF** e, consecutivamente, na **3NF**.

Pessoa (idPessoa, nome, dataNascimento)

Possíveis chaves: {idPessoa}

idPessoa → nome, dataNascimento

BCNF	✔
3NF	✔

O lado esquerdo da dependência funcional é uma *super-key*, logo esta relação encontra-se na ***BCNF*** e, consecutivamente, na ***3NF***.

Ator (idPessoa → Pessoa)

Possíveis chaves: {idPessoa}

BCNF	✔
3NF	✔

Sem dependências funcionais não triviais, logo não apresenta qualquer violação à **BCNF** e à **3NF**.

Elenco (idPessoa → PessoaAtor, idConteudo → Conteudo)

Possíveis chaves: {idPessoa, idConteudo}

BCNF	✔
3NF	✔

Sem dependências funcionais não triviais, logo não apresenta qualquer violação à **BCNF** e à **3NF**.

Realizador (idPessoa, nome, dataNascimento)

Possíveis chaves: {idPessoa}

idPessoa → nome, dataNascimento

BCNF	✔
3NF	✔

O lado esquerdo da dependência funcional é uma *super-key*, logo esta relação encontra-se na ***BCNF*** e, consecutivamente, na ***3NF***.

Realizacao (idPessoa → PessoaRealizador, idConteudo → Conteudo)

Possíveis chaves: {idPessoa, idConteudo}

BCNF	✔
3NF	✔

Sem dependências funcionais não triviais, logo não apresenta qualquer violação à **BCNF** e à **3NF**.

Pais (code_ISO, nome)

Possíveis chaves: {code_ISO}

code_ISO → nome

BCNF	✔
3NF	✔

O lado esquerdo da dependência funcional é uma *super-key*, logo esta relação encontra-se na **BCNF** e, consecutivamente, na **3NF**.

Nacionalidade (code_ISO → Pais, idPessoa → Pessoa)

Possíveis chaves: {code_ISO, idPessoa}

BCNF	✔
3NF	✔

Sem dependências funcionais não triviais, logo não apresenta qualquer violação à **BCNF** e à **3NF**.

Filme (idConteudo → Conteudo, duracao, ano)

Possíveis chaves: {idConteudo}

idConteudo → duração, ano

BCNF	✔
3NF	✔

O lado esquerdo da dependência funcional é uma *super-key*, logo esta relação encontra-se na **BCNF** e, consecutivamente, na **3NF**.

Serie (idConteudo → Conteudo, ativo)

Possíveis chaves: {idConteudo}

idConteudo → ativo

BCNF	✔
3NF	✔

O lado esquerdo da dependência funcional é uma *super-key*, logo esta relação encontra-se na **BCNF** e, consecutivamente, na **3NF**.

Temporada (idConteudo → Serie, numeroTemp, nome, ano)

Possíveis chaves: {idConteudo, numeroTemp} ou {idConteudo, nome}

idConteudo, numeroTemp → nome, ano
idConteudo, nome → numeroTemp, ano

BCNF	✔
3NF	✔

Em ambas as dependências funcionais o lado esquerdo é uma *super-key*, logo esta relação encontra-se na ***BCNF*** e consecutivamente na ***3NF***.

Episodio ([idConteudo, numeroTemp] → Temporada, numeroEp, duracao, titulo, descricao)

Possíveis chaves: {idConteudo, numeroTemp, numeroEp} ou {idConteudo, numeroTemp, titulo}

idConteudo, numeroTemp, numeroEp → duracao, titulo, descricao
idConteudo, numeroTemp, titulo → numeroEp, duracao, descrição

BCNF	✔
3NF	✔

Em ambas as dependências funcionais o lado esquerdo é uma *super-key*, logo esta relação encontra-se na ***BCNF*** e consecutivamente na ***3NF***.

VisualizouEp ([username, idCliente] → Perfil, [idConteudo, numeroTemp, numeroEp] → Episodio)

Possíveis chaves: {username, idCliente, idConteudo, numeroTemp, numeroEp}

BCNF	✔
3NF	✔

Sem dependências funcionais não triviais, logo não apresenta qualquer violação à **BCNF** e à **3NF**.

VisualizouFilme ([username, idCliente] → Perfil, idConteudo → Filme)

Possíveis chaves: {username, idCliente, idConteudo}

BCNF	✔
3NF	✔

Sem dependências funcionais não triviais, logo não apresenta qualquer violação à **BCNF** e à **3NF**.

Restrições (Entrega II)

De forma a evitar alterações não desejadas (ou a falta delas) definimos qual o comportamento sobre uma *foreign key* quando o que ela referencia é alterado (*ON UPDATE*) ou eliminado (*ON DELETE*).

- Na utilização de *ON UPDATE CASCADE* e *ON DELETE CASCADE*, as alterações efetuadas numa tabela, serão sentidas na tabela que possui a *foreign key*.
- No caso do uso de *ON UPDATE SET NULL* e *ON DELETE SET NULL*, a *foreign key* assume o valor nulo sempre que ocorra qualquer alteração ou eliminação na instância que esta referencia.
- Na eventualidade de não ser possível a alteração ou eliminação da instância para a qual *foreign key* referencia, é utilizado *ON UPDATE RESTRICT* e *ON DELETE RESTRICT*.

Cliente

Todos os clientes têm um id diferente e este não pode ser nulo:

- idCliente **PRIMARY KEY**

Todos os clientes têm de possuir um endereço de e-mail diferente

- email **UNIQUE**

Para além da *primary-key*, os seguintes atributos não podem ser nulos:

- email **NOT NULL**
- password **NOT NULL**

O credito é um atributo derivado e por omissão tem valor 0:

- credito **DEFAULT** 0

PlanoConta

Todos os planos têm um número de ecrãs diferente e não nulo, que apenas pode assumir os seguintes valores:

- ecras **PRIMARY KEY**
- ecras **CHECK** (ecras = 1 or ecras = 2 or ecras = 4)

Apenas poderão constar nesta tabela atributos nome que respeitem a seguinte restrição:

- nome **CHECK** (nome = "Base" or nome = "Standard" or nome = "Premium")

Para além disso, o preço dos planos desta tabela apenas poderão ser:

- preco **CHECK** (preco = 7.99 or preco = 10.99 or preco = 13.99)

E por fim, esta tabela possui uma restrição geral que assegura a correta combinação entre os seus atributos:

- pCorreto **CHECK** (
  (nome = "Base" and ecras = 1 and preco = 7.99)
  or (nome = "Standard" and ecras = 2 and preco = 10.99)
  or (nome = "Premium" and ecras = 4 and preco = 13.99))

Como estes elementos apenas podem constar na tabela combinados uns com os outros, apenas podemos ter uma instância de cada um deles na nossa base de dados (dado não existirem planos de conta como mesmo nome e diferentes preço ou número de ecrãs, ou ainda mesmo preço e diferentes nomes ou número de ecrãs).

Assim, todos os atributos da tabela são únicos e permitem distinguir os planos uns dos outros:

- nome **UNIQUE**
- preco **UNIQUE**

PlanoCliente

Cada cliente apenas pode subscrever a um plano por mês:

- **PRIMARY KEY** (dataInicio, idCliente)

Por omissão, dataFim é instanciado a NULL:

- dataFim **DEFAULT NULL**

A dataFim do plano de Conta tem de ser mais recente (maior) que a dataInicio do mesmo:

- **CHECK** (dataFim is NULL or dataFim > dataInicio)

Deste modo, considerando que o cliente apenas pode estar subscrito a um plano de cada vez, sendo que o atual é identificado ao ter o atributo dataFim com o valor NULL, só poderemos ter um valor de dataFim a NULL por cliente, formando uma *super-key*:

- **UNIQUE** (idCliente, dataFim)

Os atributos idCliente e ecras referenciam um Cliente e um Plano, respetivamente, sendo que nenhum deles pode ter o valor nulo:

- **FOREIGN KEY** (idCliente) **REFERENCES** Cliente(idCliente)
- **FOREIGN KEY** (ecras) **REFERENCES** PlanoConta(ecras)
- idCliente **NOT NULL**
- ecras **NOT NULL**

Para ambas as *foreign keys* é adotado um comportamento ***CASCADE*** para qualquer alteração (***ON UPDATE***) sofrida de forma a qualquer instância de PlanoCliente estar sempre atualizada.

No caso da eliminação (***ON DELETE***), é definido um comportamento ***CASCADE*** para a *foreign key* idCliente, sendo que, se o Cliente for eliminado, não valerá a pena possuir a tabela da associação do mesmo com o PlanoConta.

Por outro lado, para a *foreign key* ecras é adotado um comportamento ***RESTRICT***, de forma a não ser possível eliminar um PlanoConta no caso de existir pelo menos um cliente que está subscrito a este plano.

Pagamento

Todos os pagamentos têm de conter um id diferente e este não pode ser nulo:

- idPagamento **PRIMARY KEY**

valorUtilizadoCredito tem, por omissão, valor nulo, dado que o pagamento pode ser apenas através de multibanco e, no caso deste não ser nulo, terá que ser maior do que 0:

- valorUtilizadoCredito **DEFAULT NULL**
- **CHECK** (valorUtilizadoCredito is NULL or valorUtilizadoCredito > 0)

valorUtilizadoMultibanco tem, por omissão, valor NULL, dado que o pagamento pode ser apenas através do crédito do cliente e, no caso deste não ser NULL, tem que ser maior do que 0:

- valorUtilizadoMultibanco **DEFAULT NULL**
- **CHECK** (valorUtilizadoMultibanco is NULL or valorUtilizadoMultibanco > 0)

Para além da *primary-key*, a data também não pode ser nula, pois é um dado importante para a gestão da base de dados:

- data **NOT NULL**

PagamentoCliente

Como um pagamento apenas se refere a um e a um só cliente e um cliente pode possuir diferentes pagamentos, o que nos vai permitir diferenciar as várias associações entre um cliente e um pagamento vai ser o idPagamento que terá de ser único e não nulo e irá referenciar um pagamento:

- idPagamento **PRIMARY KEY**
- **FOREIGN KEY** (idPagamento) **REFERENCES** Pagamento(idPagamento)

Nesta relação, o atributo idCliente referencia um cliente:

- **FOREIGN KEY** (idCliente) **REFERENCES** Cliente(idCliente)

Para ambas as *foreign keys* é adotado um comportamento ***CASCADE*** para qualquer alteração (***ON UPDATE***) sofrida de forma a qualquer instância de PagamentoCliente estar sempre atualizada.

No caso da eliminação (***ON DELETE***), é definido um comportamento ***SET NULL*** para a *foreign key* idCliente, de forma a manter o registo de todos os pagamentos efetuados mesmo que as instâncias de Cliente sejam eliminadas.

Por outro lado, para a *foreign key* idPagamento é adotado um comportamento ***RESTRICT***, de forma a não ser possível eliminar uma instância de Pagamento para que exista o registo de todos os pagamentos já efetuados na base de dados.

CartaoOferta

Todos os cartões de oferta possuem um código, não nulo, que os diferencia:

- codigo **PRIMARY KEY**

Este código é constituído por um total de 11 caracteres (dígitos ou letras):

- codigo **CHECK** (length(codigo) = 11)

Apenas existem cartões físicos de oferta com os valores de 15€, 25€ e 50€, ou digitais com valores entre 25€ e 200€, e deste modo o valor cartão é restrito a:

- valor **CHECK** (valor = 15 or (valor >= 25 and valor <= 200))

Aplicaçao

Como um cliente pode aplicar vários cartões de oferta, mas estes cartões apenas podem ser associados a um só cliente, o que nos vai permitir diferenciar as várias associações entre um cliente e os seus códigos de oferta será o código dos mesmos, que referencia um cartão e não pode ser nulo:

- codigo **PRIMARY KEY**
- **FOREIGN KEY** (código) **REFERENCES** CartaoOferta(código)

A data de aplicação do cartão não pode ser nula, pois vai auxiliar a gestão do crédito do cliente:

- data **NOT NULL**

O atributo idCliente referencia o Cliente que aplicou o cartão de oferta:

- **FOREIGN KEY** (idCliente) **REFERENCES** Cliente(idCliente)

Para ambas as *foreign keys* é adotado um comportamento ***CASCADE*** para qualquer alteração (***ON UPDATE***) sofrida de forma a qualquer instância de Aplicacao estar sempre atualizada.

No caso da eliminação (***ON DELETE***), é definido um comportamento ***SET NULL*** para a *foreign key* idCliente, sendo que, se o Cliente for eliminado, continuará a existir um registo de todos os cartões de ofertas que já foram aplicados até à data.

Por outro lado, para a *foreign key* codigo é adotado um comportamento ***RESTRICT***, de forma a não ser possível eliminar uma instância de CartaoOferta no caso deste já ter sido aplicado por um Cliente (para ser possível gerir o crédito do Cliente).

CartaoMultibanco

Sabendo que não existe mais do que um cartão com o mesmo número, e que este não pode ser nulo, irá ser o numeroCartao que nos permitirá diferenciar os cartões de multibanco:

- numeroCartao **PRIMARY KEY**

Este número tem de conter 16 dígitos, sendo que o primeiro não pode ser 0 (pois irá fazer parte de uma sequência de dígitos que permitirão identificar o banco associado ao cartão):

- numeroCartao **CHECK** (numeroCartao >= 1000000000000000 and numeroCartao <= 9999999999999999)

Para além da *primary-key*, os seguintes atributos não podem ser nulos:

- dataValidade **NOT NULL**
- nome **NOT NULL**

CartaoCliente

O par (idCliente, numeroCartao) apenas é instanciado uma única vez na tabela:

- **PRIMARY KEY** (idCliente, numeroCartao)

Esta tabela relaciona um cliente com um cartão:

- **FOREIGN KEY** (idCliente) **REFERENCES** Cliente(idCliente)
- **FOREIGN KEY** (numeroCartao) **REFERENCES** CartaoMultibanco(numeroCartao)

Para ambas as *foreign keys* é adotado um comportamento ***CASCADE*** para qualquer alteração (***ON UPDATE***) ou eliminação (***ON DELETE)*** sofrida de forma a qualquer instância de CartaoPreferido estar sempre atualizada.

CartaoPreferido

Um cliente apenas pode possuir um único cartão preferido (apesar de o mesmo cartão de multibanco poder ser o cartão preferido de outros clientes). Logo, o atributo que nos permitirá diferenciar a preferência de um CartaoMultibanco por parte de um Cliente será o idCliente, que não poderá ser nulo e irá referenciar o Cliente em questão:

- idCliente **PRIMARY KEY**
- **FOREIGN KEY** (idCliente) **REFERENCES** Cliente(idCliente)

Deste modo, o CartaoMultibanco associado ao cliente será referenciado pelo seu número, e não pode ser nulo:

- numeroCartao **NOT NULL**
- **FOREIGN KEY** (numeroCartao) **REFERENCES** CartaoMultibanco(numeroCartao)

Para ambas as *foreign keys* é adotado um comportamento ***CASCADE*** para qualquer alteração (***ON UPDATE***) sofrida de forma a qualquer instância de CartaoPreferido estar sempre atualizada.

No caso da eliminação (***ON DELETE)***, é definido um comportamento ***CASCADE*** para a *foreign key* idCliente, pois não valerá a pena existir a associação entre Cliente e o seu CartaoMultibanco preferido no caso de a instância de Cliente ser eliminada.

Por outro lado, para a *foreign key* numeroCartao é adotado um comportamento ***RESTRICT***, de forma a não ser possível eliminar uma instância de CartaoMultibanco no caso deste ser o cartão preferido de algum Cliente.

Perfil

Não pode haver mais do que uma instância nesta tabela com o mesmo par (idCliente, username). De referir que um username apenas pode ter até 50 carateres:

- **PRIMARY KEY** (idCliente, username)
- username **CHECK** (lenght(username) <= 50)

Para além disso, não podem existir dois perfis do mesmo cliente com a mesma imagem de perfil:

- **UNIQUE** (idCliente, idImagem)

Cada perfil está associado a um Cliente e a uma imagem, sendo que esta não pode ser nula:

- **FOREIGN KEY** (idCliente) **REFERENCES** Cliente(idCliente)
- **FOREIGN KEY** (idImagem) **REFERENCES** Imagem(idImagem)
- idImagem **NOT NULL**

O atributo faixaEtaria não pode ser nulo e, por omissão, tem como valor '18+':

- faixaEtaria **NOT NULL**
- faixaEtaria **DEFAULT** '18+'
- faixaEtaria **CHECK** (faixaEtaria = 'ALL' **or** faixaEtaria = '7+' **or** faixaEtaria = '13+' **or** faixaEtaria = '16+' **or** faixaEtaria = '18+')

Para ambas as *foreign keys* é adotado um comportamento ***CASCADE*** para qualquer alteração (***ON UPDATE***) sofrida de forma a qualquer instância de Perfil estar sempre atualizada.

No caso da eliminação (***ON DELETE)***, é definido um comportamento ***CASCADE*** para a *foreign key* idCliente, pois não valerá a pena guardar os perfis de um Cliente no caso da instância de Cliente ser eliminada.

Por outro lado, para a *foreign key* idImagem é adotado um comportamento ***RESTRICT***, de forma a não ser possível eliminar uma instância de idImagem no caso desta estar associada a pelo menos um Perfil (pois todos os perfis têm de possuir uma e uma só Imagem).

Idioma

Diferentes idiomas têm id's diferentes (sendo que este não é nulo):

- idIdioma **PRIMARY KEY**

O atributo linguagem também não pode ser nulo:

- linguagem **NOT NULL**

Interface

Um perfil (par (username, idCliente) referenciam determinado perfil) apenas tem associado um idioma (referenciado pelo seu id) não nulo:

- **PRIMARY KEY** (username, idCliente)
- idIdioma **NOT NULL**
- **FOREIGN KEY** (idIdioma) REFERENCES Idioma(idIdioma)
- **FOREIGN KEY** (username, idCliente) **REFERENCES** Perfil(username, idCliente)

Para ambas as *foreign keys* é adotado um comportamento ***CASCADE*** para qualquer alteração (***ON UPDATE***) sofrida de forma a qualquer instância de Interface estar sempre atualizada.

No caso da eliminação (***ON DELETE)***, é definido um comportamento ***CASCADE*** para a *foreign key* referente ao Perfil, pois não valerá a manter os registos do idioma de um Perfil no caso deste ser eliminado.

Por outro lado, para a *foreign key* idIdioma é adotado um comportamento ***RESTRICT***, de forma a não ser possível eliminar uma instância de idIdioma no caso desta estar associada a pelo menos um Perfil (pois todos os perfis têm de possuir um e um só idioma associado).

Caracteristica

As características podem ser diferenciadas pelo seu id e pelo seu nome e, deste modo, estes atributos têm de ser únicos e não nulos:

- idCaracteristica **PRIMARY KEY**
- nome **UNIQUE**
- nome **NOT NULL**

CaracteristicaConteudo

Não pode existir mais do que uma instância do par (idCaracteristica, idConteudo), pois uma determinada característica apenas pode ser associada a um determinado conteúdo uma única vez:

- **PRIMARY KEY** (idCaracteristica, idConteudo)

Deste modo, os atributos idCaracteristica e idConteudo referenciam, respetivamente, uma Caracteristica e um Conteudo:

- **FOREIGN KEY** (idCaracteristica) **REFERENCES** Caracteristica (idCaracteristica)
- **FOREIGN KEY** (idConteudo) **REFERENCES** Conteudo(idConteudo)

Para ambas as *foreign keys* é adotado um comportamento ***CASCADE*** para qualquer alteração (***ON UPDATE***) sofrida de forma a qualquer instância de CaracteristicaConteudo estar sempre atualizada.

No caso da eliminação (***ON DELETE***), é definido um comportamento ***CASCADE*** para a *foreign key* idConteudo, pois não valerá a pena existir a associação entre Conteudo as suas características no caso da instância de Conteudo ser eliminada.

Por outro lado, para a *foreign key* idCaracteristica é adotado um comportamento ***RESTRICT***, de forma a não ser possível eliminar uma instância de Caracteristica no caso desta estar associada a algum Conteudo.

Genero

Os géneros podem ser diferenciados pelo seu id e pelo seu nome, e, deste modo, estes atributos têm de ser únicos e não nulos:

- idGenero **PRIMARY KEY**
- nome **UNIQUE**
- nome **NOT NULL**

GeneroConteudo

Não pode existir mais do que uma instância do par (idGenero, idConteudo), pois um determinado género apenas pode ser associado a um determinado conteúdo uma única vez:

- **PRIMARY KEY** (idGenero, idConteudo)

Deste modo, os atributos idGenero e idConteudo referenciam, respetivamente, um Genero e um Conteudo:

- **FOREIGN KEY** (idGenero) **REFERENCES** Genero (idGenero)
- **FOREIGN KEY** (idConteudo) **REFERENCES** Conteudo(idConteudo)

Para ambas as *foreign keys* é adotado um comportamento ***CASCADE*** para qualquer alteração (***ON UPDATE***) sofrida de forma a qualquer instância de GeneroConteudo estar sempre atualizada.

No caso da eliminação (***ON DELETE***), é definido um comportamento ***CASCADE*** para a *foreign key* idConteudo, pois não valerá a pena existir a associação entre Conteudo os seus géneros no caso da instância de Conteudo ser eliminada.

Por outro lado, para a *foreign key* idGenero é adotado um comportamento ***RESTRICT***, de forma a não ser possível eliminar uma instância de Genero no caso desta estar associada a algum Conteudo.

Imagem

As imagens serão diferenciadas pelo seu id não nulo e possuirão um *pathImagem* único que também não pode ser nulo:

- idImagem **PRIMARY KEY**
- pathImagem **UNIQUE**
- pathImagem **NOT NULL**

ImagemConteudo

Uma imagem não pode estar associada a mais do que um conteúdo. Deste modo, o atributo idImagem, que referencia uma Imagem, será a chave primária desta tabela (dado que um conteúdo, referenciado pelo atributo idConteudo não nulo, pode conter mais do que uma imagem associada):

- idImagem **PRIMARY KEY**
- idConteudo **NOT NULL**
- **FOREIGN KEY** (idImagem) **REFERENCES** Imagem(idImagem)
- **FOREIGN KEY** (idConteudo) **REFERENCES** Conteudo(idConteudo)

Para ambas as *foreign keys* é adotado um comportamento ***CASCADE*** para qualquer alteração (***ON UPDATE***) sofrida de forma a qualquer instância de ImagemConteudo estar sempre atualizada.

No caso da eliminação (***ON DELETE***), é definido um comportamento ***CASCADE*** para a *foreign key* idConteudo, pois não valerá a pena existir a associação entre Conteudo as suas imagens no caso da instância de Conteudo ser eliminada.

Por outro lado, para a *foreign key* idImagem é adotado um comportamento ***RESTRICT***, de forma a não ser possível eliminar uma instância de Imagem no caso desta estar associada a algum Conteudo.

ImagemEpisodio

Como pode haver a possibilidade de uma imagem estar associada a um ou mais episódios, mas cada um destes só podem ter uma imagem, a chave primária desta tabela irá ser constituída pelos atributos idConteudo, numeroTemp e numeroEp que irão referenciar um Episodio associando-o a uma Imagem (referenciada pelo atributo idImagem):

- **PRIMARY KEY** (idConteudo, numeroTemp, numeroEp)
- **FOREIGN KEY** (idImagem) REFERENCES Imagem(idImagem)
- **FOREIGN KEY** (idConteudo, numeroTemp, numeroEp) **REFERENCES** Episodio(idConteudo, numeroTemp, numeroEp)

Conteudo

Os conteúdos são diferenciados pelos seus id's não nulos:

- idConteudo **PRIMARY KEY**

O atributo classificacao é um número real entre 0.0 e 5.0 inclusive:

- classificacao **CHECK** (classificacao >= 0.0 and classificacao <= 5.0)

Para além disto, os seguintes atributos não podem ser nulos:

- nome **NOT NULL**
- faixaEtariaMinima **NOT NULL**, por omissão: faixaEtariaMinima **DEFAULT** '18+' e apenas pode assumir: faixaEtariaMinima **CHECK** (faixaEtariaMinima = 'ALL' **or** faixaEtariaMinima = '7+' **or** faixaEtariaMinima = '13+' **or** faixaEtariaMinima = '16+' **or** faixaEtariaMinima = '18+')
- descricao **NOT NULL**

Pessoa

Todas as pessoas têm de ter um nome e uma data de nascimento, não nulos, e podem ser diferenciadas pelos seus id's que também não podem ser nulos:

- idPessoa **PRIMARY KEY**
- nome **NOT NULL**
- dataNascimento **NOT NULL**

Ator

Uma pessoa pode exercer a profissão de Ator, e, deste modo, teremos nesta tabela um único atributo idPessoa que irá referenciar uma Pessoa que exerce tal profissão:

- idPessoa **PRIMARY KEY**
- **FOREIGN KEY** (idPessoa) **REFERENCES** Pessoa(idPessoa)

Para a *foreign key* é adotado um comportamento ***CASCADE*** para qualquer alteração (***ON UPDATE***) ou eliminação (***ON DELETE)*** sofrida de forma a qualquer instância de Ator estar sempre atualizada.

Elenco

Não poderá existir mais do que uma instância do par (idPessoa, idConteudo) nesta tabela:

- **PRIMARY KEY** (idPessoa, idConteudo)

Os atributos idPessoa e idConteudo referenciam um Ator que participa num certo Conteudo:

- **FOREIGN KEY** (idPessoa) **REFERENCES** Ator(idPessoa)
- **FOREIGN KEY** (idConteudo) **REFERENCES** Conteudo(idConteudo)

Para ambas as *foreign keys* é adotado um comportamento ***CASCADE*** para qualquer alteração (***ON UPDATE***) sofrida de forma a qualquer instância de Elenco estar sempre atualizada.

No caso da eliminação (***ON DELETE***), é definido um comportamento ***CASCADE*** para a *foreign key* idConteudo, pois não valerá a pena existir a associação entre Conteudo e os seus atores no caso da instância de Conteudo ser eliminada.

Por outro lado, para a *foreign key* idPessoa é adotado um comportamento ***RESTRICT***, de forma a não ser possível eliminar uma instância de Ator no caso deste estar associada a algum Conteudo.

Realizador

Uma pessoa pode exercer a profissão de Realizador, e, deste modo, teremos nesta tabela um único atributo idPessoa que irá referenciar uma Pessoa que exerce tal profissão:

- idPessoa **PRIMARY KEY**
- **FOREIGN KEY** (idPessoa) **REFERENCES** Pessoa(idPessoa)

Para a *foreign key* é adotado um comportamento ***CASCADE*** para qualquer alteração (***ON UPDATE***) ou eliminação (***ON DELETE***) sofrida de forma a qualquer instância de Realizador estar sempre atualizada.

Realizacao

Não poderá haver mais do que uma instância do par (idPessoa, idConteudo) nesta tabela:

- **PRIMARY KEY** (idPessoa, idConteudo)

Os atributos idPessoa e idConteudo referenciam um Realizador que produziu um certo Conteudo:

- **FOREIGN KEY** (idPessoa) **REFERENCES** Ator(idPessoa)
- **FOREIGN KEY** (idConteudo) **REFERENCES** Conteudo(idConteudo)

Para ambas as *foreign keys* é adotado um comportamento ***CASCADE*** para qualquer alteração (***ON UPDATE***) sofrida de forma a qualquer instância de Realizacao estar sempre atualizada.

No caso da eliminação (***ON DELETE***), é definido um comportamento ***CASCADE*** para a *foreign key* idConteudo, pois não valerá a pena existir a associação entre Conteudo e os seus realizadores no caso da instância de Conteudo ser eliminada.

Por outro lado, para a *foreign key* idPessoa é adotado um comportamento ***RESTRICT***, de forma a não ser possível eliminar uma instância de Realizador no caso deste estar associada a algum Conteudo.

Pais

Todos os países possuem um *code_ISO* diferente e este não pode ser nulo:

- code_ISO **PRIMARY KEY**

Para além disto, o nome do país também não pode ser nulo:

- nome **NOT NULL**

Nacionalidade

Não poderá haver mais do que uma instância do par (code_ISO, idPessoa) nesta tabela:

- **PRIMARY KEY** (code_ISO, idPessoa)

Os atributos code_ISO e idPessoa referenciam um Pais e uma Pessoa respetivamente:

- **FOREIGN KEY** (code_ISO) **REFERENCES** Pais(code_ISO)
- **FOREIGN KEY** (idPessoa) **REFERENCES** Pessoa(idPessoa)

Para ambas as *foreign keys* é adotado um comportamento ***CASCADE*** para qualquer alteração (***ON UPDATE***) sofrida de forma a qualquer instância de Nacionalidade estar sempre atualizada.

No caso da eliminação (***ON DELETE)***, é definido um comportamento ***CASCADE*** para a *foreign key* idPessoa, pois não valerá a pena existir o registo das nacionalidades de uma Pessoa no caso da mesma ser eliminada da base de dados.

Por outro lado, para a *foreign key* code_ISO é adotado um comportamento ***RESTRICT***, de forma a não ser possível eliminar uma instância de Pais no caso deste estar associada a alguma Pessoa.

Filme

Todos os filmes são um conteúdo e, por isso, têm um idConteudo não nulo (que irá referenciar qual o Conteudo associado a cada elemento da tabela) que os permite diferenciar:

- idConteudo **PRIMARY KEY**
- **FOREIGN KEY** (idConteudo) Conteudo(idConteudo)

Nenhum filme pode ter uma duração negativa ou igual a 0:

- **CHECK** (duracao > 0)

Para além disso, cada filme tem sempre um ano associado:

- ano **NOT NULL**

Para a *foreign key* é adotado um comportamento ***CASCADE*** para qualquer alteração (***ON UPDATE***) ou eliminação (***ON DELETE)*** sofrida de forma a qualquer instância de Filme estar sempre atualizada.

Serie

Todas as séries são um conteúdo e, por isso, têm um idConteudo não nulo (que irá referenciar qual o Conteudo associado a cada elemento da tabela) que as permite diferenciar:

- idConteudo **PRIMARY KEY**
- **FOREIGN KEY** (idConteudo) Conteudo(idConteudo)

Para além disso, cada série terá um atributo não nulo que indica se esta está ativa (ou seja, se ainda terá futuras temporadas a serem acrescentadas) ou não:

- ativo **NOT NULL**

Para a *foreign key* é adotado um comportamento ***CASCADE*** para qualquer alteração (***ON UPDATE***) ou eliminação (***ON DELETE)*** sofrida de forma a qualquer instância de Série estar sempre atualizada.

Temporada

Não pode existir mais do que uma instância igual do par (idConteudo e numeroTemp), dado que para um determinado Conteudo (Serie), referenciado por idConteudo, não existem temporadas com o mesmo número:

- **PRIMARY KEY** (idConteudo, numeroTemp)
- **FOREIGN KEY** (idConteudo) **REFERENCES** Serie(idConteudo)

Numa série, também não podem existir duas temporadas com o mesmo nome:

- **UNIQUE** (idConteudo, nome)

O atributo numeroTemp tem de ser maior que 0:

- **CHECK** (numeroTemp > 0)

Para além dos já referidos, os seguintes atributos também não podem ser nulos:

- nome **NOT NULL**
- ano **NOT NULL**

Para a *foreign key* é adotado um comportamento ***CASCADE*** para qualquer alteração (***ON UPDATE***) ou eliminação (***ON DELETE)*** sofrida de forma a qualquer instância de Temporada estar sempre atualizada.

Episodio

Não pode existir mais do que uma instância igual do conjunto (idConteudo, numeroTemp e numeroEp), dado que para uma determinada Temporada (referenciada por idConteudo e numeroTemp) não existem episódios com o mesmo número:

- **PRIMARY KEY** (idConteudo, numeroTemp, numeroEp)
- **FOREIGN KEY** (idConteudo, numeroTemp) **REFERENCES** Temporada(idConteudo, numeroTemp)

Para além disso, também não podem existir episódios da mesma temporada com o mesmo título:

- **UNIQUE** (idConteudo, numeroTemp, titulo)

Os atributos numeroEp e duracao têm de ser maiores que 0:

- **CHECK** (numeroEp > 0)
- **CHECK** (duracao > 0)

Para além disso, os seguintes atributos não podem ser nulos:

- titulo **NOT NULL**
- descricao **NOT NULL**

A duracao é um número real, sendo a parte inteira correspondente aos minutos e a parte decimal aos segundos.

Para a *foreign key* é adotado um comportamento ***CASCADE*** para qualquer alteração (***ON UPDATE***) ou eliminação (***ON DELETE)*** sofrida de forma a qualquer instância de Episodio estar sempre atualizada.

VisualizouEp

Não pode existir mais do que uma instância do grupo (username, idCliente, idConteudo, numeroTemp, numeroEp), identificando que um determinado Perfil de um Cliente (referenciado pelo par (username, idCliente)), visualizou um determinado episódio (referenciado pelo conjunto (idConteudo, numeroTemp, numeroEp)):

- **PRIMARY KEY** (username, idCliente, idConteudo, numeroTemp, numeroEp)
- **FOREIGN KEY** (username, idCliente) **REFERENCES** Perfil(username, idCliente)
- **FOREIGN KEY** (idConteudo, numeroTemp, numeroEp) **REFERENCES** Episodio(idConteudo, numeroTemp, numeroEp)

Para ambas as *foreign keys* é adotado um comportamento ***CASCADE*** para qualquer alteração (***ON UPDATE***) sofrida de forma a qualquer instância de VisualizouEp estar sempre atualizada.

No caso da eliminação (***ON DELETE)***, é definido um comportamento ***SET NULL*** para ambas as *foreign keys*, de forma ser possível contabilizar quantos episódios um Perfil já viu, mesmo que alguns já tenham sido apagados, e controlar quantos perfis já viram um determinado Episódio, mesmo que alguns destes já tenham sido eliminados.

VisualizouFilme

Não pode existir mais do que uma instância do grupo (username, idCliente, idConteudo), identificando que um determinado Perfil de um Cliente (referenciado pelo par (username, idCliente)), visualizou um determinado Filme (referenciado pelo atributo idConteudo):

- **PRIMARY KEY** (username, idCliente, idConteudo)
- **FOREIGN KEY** (username, idCliente) REFERENCES Perfil(username, idCliente)
- **FOREIGN KEY** (idConteudo) **REFERENCES** Filme(idConteudo)

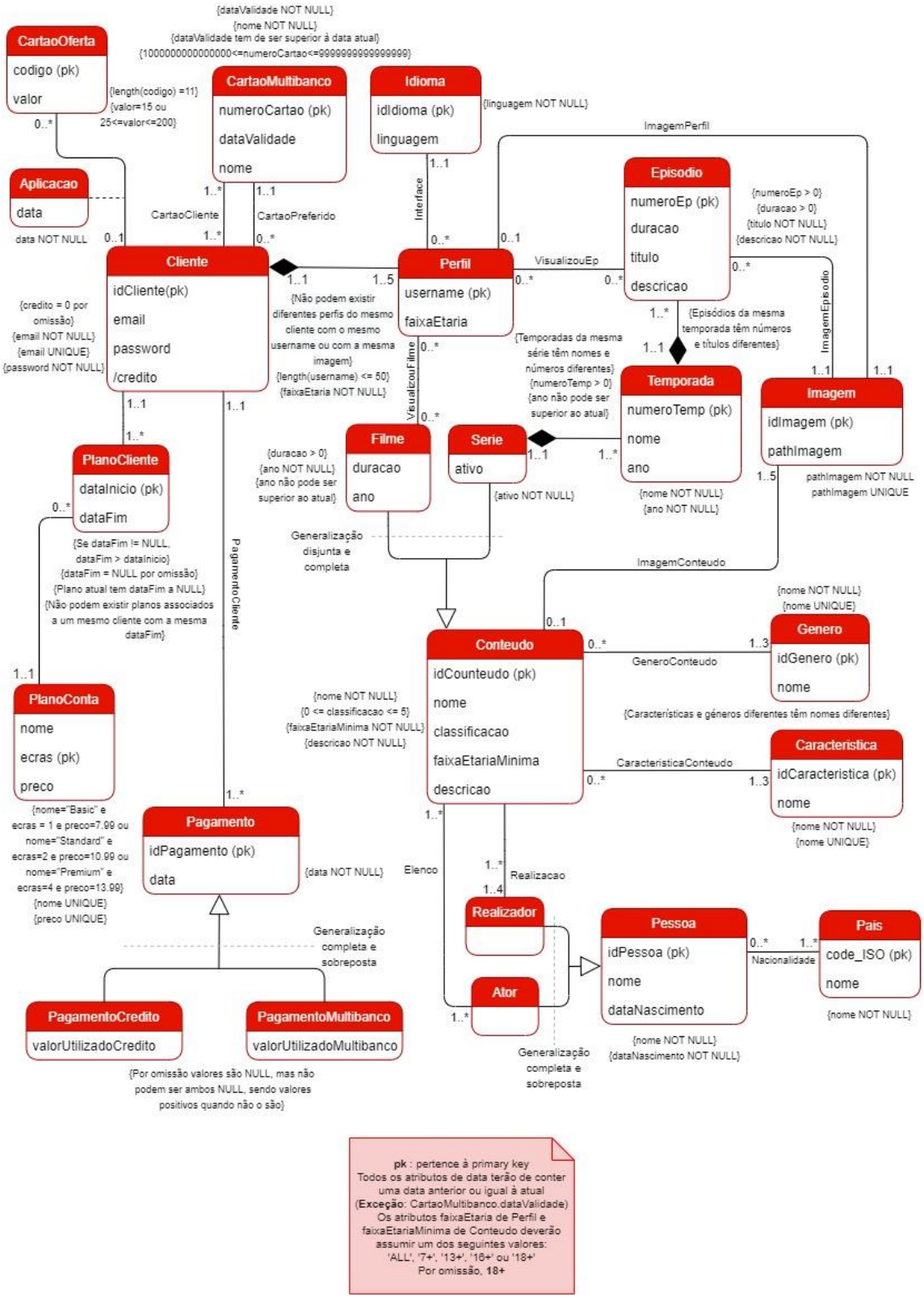
Para ambas as *foreign keys* é adotado um comportamento ***CASCADE*** para qualquer alteração (***ON UPDATE***) sofrida de forma a qualquer instância de VisualizouFilme estar sempre atualizada.

No caso da eliminação (***ON DELETE)***, é definido um comportamento ***SET NULL*** para ambas as *foreign keys*, de forma ser possível contabilizar quantos filmes um Perfil já viu, mesmo que alguns já tenham sido apagados, e controlar quantos perfis já viram um determinado Filme, mesmo que alguns destes já tenham sido eliminados.

Nota:

Todas as restrições apresentadas no Modelo Conceptual que envolvam a data atual, não foram adicionadas por ainda não possuirmos conhecimento suficiente para as implementar.

Novo diagrama UML



Ana Matilde Guedes Perez da Silva Barra up201904795
Maria Beatriz Russo Lopes dos Santos up201906888
Tiago Caldas da Silva up201906045

Lista de Interrogações

1. Caso um Perfil tenha começado a visualizar uma série, indicar a percentagem que falta para acabar a respetiva.

2. Listar todos os conteúdos que contenham uma determinada característica em comum.

3. Média de dinheiro ganho até à data atual por diferentes meios: apenas por cartão de oferta, apenas por cartão de multibanco, junção dos dois referidos.

4. Conteúdo mais visto.

5. Perfil que visualizou mais conteúdo, indicando os respetivos minutos de visualização.

6. Para cada conteúdo, indicar o número de perfis de cada faixa etária que o visualizou

7. Perfis de Clientes que visualizaram todos os episódios de uma série.

8. Para cada Cliente, caso exista, indicar os Perfis que visualizaram o mesmo Conteúdo identificado pelo seu nome.

9. Listar o nome de conteúdos que começam com a letra "M", no caso das séries basta ter um episódio a começar com essa letra e ordenar por número de visualizações.

10. Cliente que passou mais tempo com a sua subscrição na *Netflix* em *stand-by* (sem pagar mensalidade), indicando o número de dias.

Nota:

De forma a dificultar a interrogação número 4, foi evitado o uso de funcionalidades como ***max*** e ***limit***.

De forma a simular a ferramenta de pesquisa da *Netflix*, na interrogação número 9 consideramos tanto o uso de "M" como de "m".

Ana Matilde Guedes Perez da Silva Barra up201904795
Maria Beatriz Russo Lopes dos Santos up201906888
Tiago Caldas da Silva up201906045

Lista de Gatilhos

1. Verificar se uma visualização inserida é válida, ou seja, verificar se um perfil possui uma faixa etária maior ou igual à permitida para o conteúdo a visualizar.

2. Verificar se um pagamento inserido é valido.

3. Aumentar o valor de crédito de um cliente após a aplicação de um cartão oferta.

Ana Matilde Guedes Perez da Silva Barra up201904795
Maria Beatriz Russo Lopes dos Santos up201906888
Tiago Caldas da Silva up201906045