
BDAD PROJECT

2020

RELATÓRIO (ENTREGA 3)

MIEIC07 – Grupo 705

Eduardo Brito - up201806271

Pedro Ferreira - up201806506

Pedro Ponte - up201809694

WORKFLOW

Página <u>3</u>		Modelo Conceptual Revisão
<u>6</u>		UML Classes e Atributos
<u>8</u>		Modelo Relacional Contextualização
<u>10</u>		Esquema e Dependências Funcionais
<u>13</u>		Cálculo das Chaves Candidatas
<u>15</u>		Análise das Dependências Funcionais e Formas Normais
<u>16</u>		Restrições
<u>21</u>		Interrogações
<u>22</u>		Gatilhos
		Entrega III

Modelo Conceptual

Revisão

Este projeto baseia-se na gestão e funcionamento de uma loja virtual de videojogos.

Numa visão global do contexto associado ao tema deste projeto, é possível identificar três focos conceptuais de destaque, no modelo criado.

O primeiro é relativo à classe **Produto**, que representa a generalização das suas subclasses. Esta generalização, estabelecendo uma associação com a classe **Marca** é *Completa e Disjunta*, sendo que um **Produto** pode apenas existir como uma **Consola**, um **Acessório**, um **Videojogo**, ou um artigo de **Merchandising**.

Além dos atributos herdados na generalização e comuns a todos os produtos, nomeadamente, o *SKU* (código de identificação comercial do produto), o *Nome*, o *Preço Base* e o *Stock* (quantidade existente para venda), todas estas subclasses têm os seus atributos próprios. Convém notar que um **Acessório**, ou um **Videojogo**, terá de ser compatível com certa(s) **Consola(s)**.

O segundo foco é relativo à classe **Cliente**, cuja informação é guardada através do seu *Username*, *Nome*, *Email*, *Data de Nascimento*, *Morada e Código Postal*, *Data de Criação da Conta*, e o número total de *Pontos* acumulados.

Por uma questão de marketing, um **Cliente** tem a possibilidade de avaliar criticamente os videojogos, sendo que o *Valor* atribuído a essa **Avaliação** contribui funcionalmente para o *Rating* final de um dado **Videojogo**.

Um **Cliente** tem a si associada uma **Localização** genérica, global, continental, ou até por país, que poderá servir para, eventualmente, criar um ranking de clientes, baseado nos seus *Pontos*.

Estes têm ainda um **Estatuto** identificativo, com utilização e motivo semelhantes à **Localização**, definido por patamares (*Mínimo e Máximo de Pontos*). Exemplos de possíveis estatutos são *Newbie, Experienced, Premium, Platinum...*

Um **Cliente** tem a possibilidade de alcançar conquistas virtuais, genéricas (sendo, por isso, uma generalização *Incompleta e Disjunta*), ou conquistas em função do seu tempo de **Fidelização**, do seu **Nível de Pontos**, ou até do **Número de Compras** efetuadas. Cada **Conquista** tem um número de *Pontos* associado, funcionando como prémio para o **Cliente**.

Por último, existe a classe **Compra**, efetuada por um **Cliente**, com o respetivo *Valor, Data e Hora*, e número de *Pontos* ganho (em função do *Valor*).

Cada **Compra** está associada a produtos ou bundles, em **Quantidade** variável, que influenciará a atualização do *Stock*. Um **Bundle** não é mais do que um conjunto de produtos vendidos em pacote. Nestas associações é importante realçar a seguinte restrição: uma **Compra** não pode ser vazia, isto é, tem de estar associada a pelo menos um **Produto** e/ou **Bundle**.

Além disto, uma **Compra** pode, também, estar associada a descontos. Esta última generalização é *Completa e Disjunta*, já que só existirão descontos de três tipos: **Desconto Por Pontos**, aplicado em função dos *Pontos* do **Cliente** que realiza a **Compra**; **Desconto Sazonal**, com uma *Data de Início e Data de Fim*; e **Desconto Especial**, também com datas definidas, mas aplicado a determinados produtos.

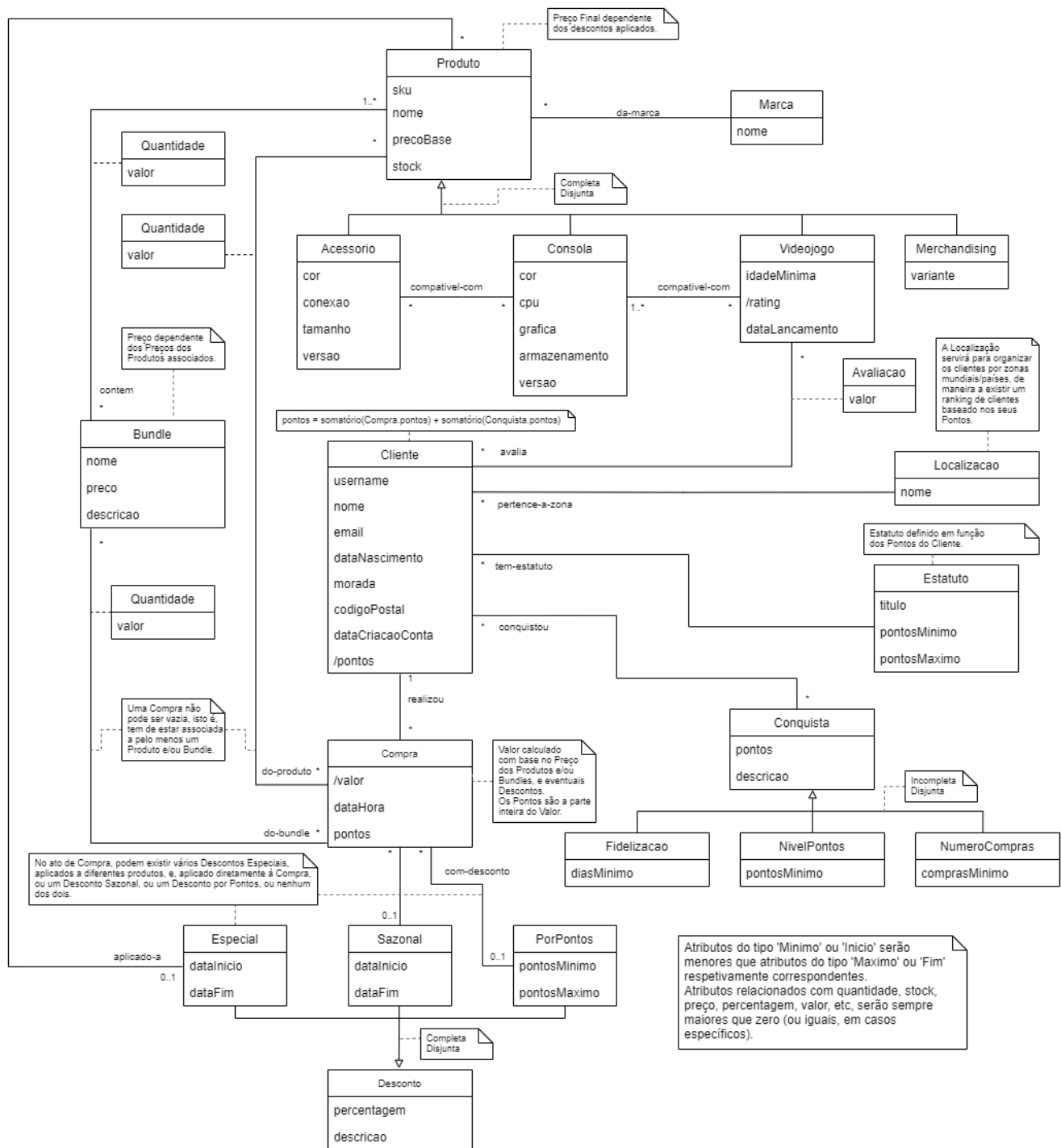
Existem, aqui, várias particularidades a mencionar. Um **Desconto** tem sempre uma dada *Percentagem* a aplicar, que influencia o *Valor* final da **Compra**. No entanto, um **Desconto por Pontos**, ou **Desconto Sazonal**, aplica a sua percentagem diretamente ao *Valor* da **Compra**, ao contrário de um **Desconto Especial** que é aplicado ao *Preço* de um **Produto** específico, acabando, mas de outra forma, por influenciar, igualmente, o *Valor* final a pagar. No ato de **Compra**, podem existir vários **Descontos Especiais**, aplicados a diferentes produtos, e, ou um **Desconto Sazonal**, ou um **Desconto por Pontos**, ou nenhum dos dois.

Como última nota, é importante referir que os *Pontos* de um **Cliente** serão sempre acumulados, uma vez que serão calculados pelo somatório de todos os *Pontos* das Compras efetuadas e de todos os *Pontos* ganhos com as Conquistas, permitindo, assim, estabelecer os patamares dos Estatutos, das Conquistas e dos Descontos por Pontos eventualmente existentes.

Segue-se o modelo conceptual, criado* em UML, que define graficamente e com maior detalhe a organização de todas as Classes e respetivos atributos, que, posteriormente, figurarão na base de dados a criar. Neste modelo estão indicadas várias restrições necessárias, as multiplicidades das associações e as generalizações anteriormente mencionadas.

*** O Modelo conceptual foi revisto e alterado, conforme o feedback da Entrega I.**

UML



Classes e Atributos

Produto

sku
nome
precoBase
stock

Marca

nome

Console

cor
cpu
grafica
armazenamento
versao

Acessorio

cor
conexao
tamanho
versao

Videojogo

idadeMinima
/rating
dataLancamento

Merchandising

Variante

Quantidade

valor

Bundle

nome
preco
descricao

Compra

/valor
dataHora
pontos

Cliente

username
nome
email
dataNascimento
morada
codigoPostal
dataCriacaoConta
/pontos

Localizacao

nome

Avaliacao

valor

Estatuto

titulo
pontosMinimo
pontosMaximo

Conquista

pontos
descricao

Fidelizacao

diasMinimo

NivelPontos

pontosMinimo

NumeroCompras

comprasMinimo

Desconto

percentagem
descricao

Especial

dataInicio
dataFim

Sazonal

dataInicio
dataFim

PorPontos

pontosMinimo
pontosMaximo

Modelo Relacional

Contextualização

Aquando do mapeamento do modelo conceitual para o esquema relacional, várias opções divergentes surgiram. Existia a possibilidade de adotar diferentes estratégias para a representação das generalizações presentes no esquema, o que iria, eventualmente, conduzir a resultados também distintos. As opções pendiam sobre o estilo *Object Oriented* e o estilo *E/R*, nas generalizações relacionadas com as classes **Produto** e **Conquista**. A primeira opção resultava num esquema relacional em que as classes que sobravam dessa generalização congregavam um número maior de atributos. Assim, em algumas relações que resultavam de associações com as anteriores, forçar-se-ia o uso de atributos com valor *NULL*, para mitigar as consequências da escolha inicial.

Um exemplo que demonstraria o relatado é o seguinte:

Acessorio (*sku, nome, precoBase, stock, marca* → *Marca, cor, conexao, tamanho, versao, idDesconto* → *DescontoEspecial*)

Consola (*sku, nome, precoBase, stock, marca* → *Marca, cor, cpu, grafica, armazenamento, versao, idDesconto* → *DescontoEspecial*)

Videojogo (*sku, nome, precoBase, stock, marca* → *Marca, idadeMinima, rating, dataLancamento, idDesconto* → *DescontoEspecial*)

Merchandising (*sku, nome, precoBase, stock, marca* → *Marca, variante, idDesconto* → *DescontoEspecial*)

...

Bcontemp (*idB* → *Bundle, skuA* → *Acessorio, skuC* → *Consola, skuV* → *Videojogo, skuM* → *Merchandising, quantidade*)

...

CompraDoProduto (*id, idCompra* → *Compra, skuA* → *Acessorio, skuC* → *Consola, skuV* → *Videojogo, skuM* → *Merchandising, quantidade*)

A adoção do estilo *Object Oriented* implicaria a passagem de todos os atributos da relação **Produto** para as relações subsequentes. Além disso, como, por exemplo, a

relação **BcontemP**, que materializa a pertença de determinados **Produtos** a um **Bundle**, necessitaria de manter a integridade referencial através de chaves estrangeiras, mas, ao mesmo tempo, só deveria referenciar um **Produto** de cada vez, com uma certa quantidade associada, vários dos seus atributos iriam ser *NULL*, por exclusão mútua. Aconteceria o mesmo na relação **CompraDoProduto**, que resultou da classe de associação que ligava uma **Compra** a um **Produto**.

Para ultrapassar as desvantagens desse estilo, nestes casos, foi adotado o *E/R style*, mantendo as classes genéricas e criando as relações derivadas, com menos atributos e sem recurso a *NULLs*. No caso restante, da generalização associada à classe Desconto, o estilo seguido foi o *Object Oriented*.

Baseada nas restantes diretivas, expostas ao longo das aulas teóricas, sobre o mapeamento do modelo conceptual para o modelo relacional, segue-se, então, a efetiva materialização de todo o esquema relacional e das dependências funcionais.

Esquema Relacional e Dependências Funcionais

Marca (id, nome)

1. $id \rightarrow nome$
2. $nome \rightarrow id$

Produto (sku, nome, precoBase, stock, marca \rightarrow Marca, idDesconto \rightarrow DescontoEspecial)

1. $sku \rightarrow nome, precoBase, stock, marca, idDesconto$
2. $nome, marca \rightarrow sku$

Acessorio (sku \rightarrow Produto, cor, conexao, tamanho, versao)

1. $sku \rightarrow cor, conexao, tamanho, versao$

Consola (sku \rightarrow Produto, cor, cpu, grafica, armazenamento, versao)

1. $sku \rightarrow cor, cpu, grafica, armazenamento, versao$

Videojogo (sku \rightarrow Produto, idadeMinima, rating, dataLancamento)

1. $sku \rightarrow idadeMinima, rating, dataLancamento$

Merchandising (sku \rightarrow Produto, variante)

1. $sku \rightarrow variante$

AcompativelComC (skuA \rightarrow Acessorio, skuC \rightarrow Consola)

Sem dependências funcionais não triviais.

VcompativelComC (skuV \rightarrow Videojogo, skuC \rightarrow Consola)

Sem dependências funcionais não triviais.

Bundle (id, nome, preco, descricao)

1. $id \rightarrow nome, preco, descricao$

BcontemP (idB → Bundle, skuP → Produto, quantidade)

1. idB, skuP → quantidade

Cliente (id, username, nome, email, dataNascimento, morada, codigoPostal, dataCriacaoConta, pontos, localizacao → Localizacao, estatuto → Estatuto)

1. id → username, nome, email, dataNascimento, morada, codigoPostal, dataCriacaoConta, pontos, localizacao, estatuto
2. username → id
3. email → id

Avaliacao (idCliente → Cliente, skuV → Videojogo, valor)

1. idCliente, skuV → valor

Localizacao (id, nome)

1. id → nome
2. nome → id

Estatuto (id, titulo, pontosMinimo, pontosMaximo)

1. id → titulo, pontosMinimo, pontosMaximo
2. titulo → id

Conquista (id, pontos, descricao)

1. id → pontos, descricao

Fidelizacao (id → Conquista, diasMinimo)

1. id → diasMinimo

NivelPontos (id → Conquista, pontosMinimo)

1. id → pontosMinimo

NumeroCompras (id → Conquista, comprasMinimo)

1. id → comprasMinimo

Conquistou (idCliente → Cliente, idConquista → Conquista)

Sem dependências funcionais não triviais.

Compra (id, idCliente → Cliente, valor, dataHora, pontos, idSazonal → DescontoSazonal, idPorPontos → DescontoPorPontos)

1. id → idCliente, valor, dataHora, pontos, idSazonal, idPorPontos
2. idCliente, dataHora → id

CompraDoBundle (idCompra → Compra, idBundle → Bundle, quantidade)

1. idCompra, idBundle → quantidade

CompraDoProduto (idCompra → Compra, idProduto → Produto, quantidade)

1. idCompra, idProduto → quantidade

DescontoEspecial (id, percentagem, descricao, dataInicio, dataFim)

1. id → percentagem, descrição, dataInicio, dataFim
2. percentagem, dataInicio, dataFim → id

DescontoSazonal (id, percentagem, descricao, dataInicio, dataFim)

1. id → percentagem, descrição, dataInicio, dataFim
2. percentagem, dataInicio, dataFim → id

DescontoPorPontos (id, percentagem, descricao, pontosMinimo, pontosMaximo)

1. id → percentagem, descrição, pontosMinimo, pontosMaximo
2. percentagem, pontosMinimo, pontosMaximo → id

Cálculo das Chaves Candidatas

O cálculo das Chaves Candidatas e estabelecimento de uma Chave Primária para cada relação (já identificadas na secção anterior), considerando apenas as dependências funcionais não triviais presentes no modelo relacional, efetuou-se da seguinte forma:

Marca:

1. $\{id\}^+ = \{id, nome\}$
 2. $\{nome\}^+ = \{id, nome\}$
- id é a chave primária.

Produto:

1. $\{sku\}^+ = \{sku, nome, precoBase, stock, marca, idDesconto\}$
 2. $\{nome, marca\}^+ = \{sku, nome, precoBase, stock, marca, idDesconto\}$
- sku é a chave primária.

Acessorio:

1. $\{sku\}^+ = \{sku, cor, conexao, tamanho, versao\}$
- sku é a chave primária.

Semelhante para as relações Consola, Videojogo e Merchandising.

AcompativelComC:

1. $\{skuA, skuC\}^+ = \{skuA, skuC\}$
- $\{skuA, skuC\}$ é a chave primária.

Semelhante para as relações VcompativelComC, BcontemP, Avaliacao, Conquistou, CompraDoBundle e CompraDoProduto.

Bundle:

1. $\{id\}^+ = \{id, nome, preco, descricao\}$
- id é a chave primária.

Cliente:

1. {id}⁺ = {id, username, nome, email, dataNascimento, morada, codigoPostal, dataCriacaoConta, pontos, localizacao, estatuto}
2. {username}⁺ = {id, username, nome, email, dataNascimento, morada, codigoPostal, dataCriacaoConta, pontos, localizacao, estatuto}
3. {email}⁺ = {id, username, nome, email, dataNascimento, morada, codigoPostal, dataCriacaoConta, pontos, localizacao, estatuto}

id é a chave primária.

Localizacao:

1. {id}⁺ = {id, nome}
2. {nome}⁺ = {id, nome}

id é a chave primária.

Estatuto:

1. {id}⁺ = {id, titulo, pontosMinimo, pontosMaximo}
2. {titulo}⁺ = {id, titulo, pontosMinimo, pontosMaximo}

id é a chave primária.

Conquista:

1. {id}⁺ = {id, pontos, descricao}

id é a chave primária.

Semelhante para as relações Fidelizacao, NivelPontos e NumeroCompras.

Compra:

1. {id}⁺ = {id, idCliente, valor, dataHora, pontos, idSazonal, idPorPontos}
2. {idCliente, dataHora}⁺ = {id, idCliente, valor, dataHora, pontos, idSazonal, idPorPontos}

id é a chave primária.

DescontoEspecial:

1. {id}⁺ = {id, percentagem, descricao, dataInicio, dataFim}

2. {percentagem, dataInicio, dataFim}⁺ = {id, percentagem, descricao, dataInicio, dataFim}

id é a chave primária.

Semelhante para as relações DescontoSazonal e DescontoPorPontos.

Análise das Dependências Funcionais e Formas Normais

Efetuada o cálculo das chaves candidatas de cada relação, averiguámos possíveis violações às formas normais e constatámos que, em todas as dependências funcionais não triviais, o lado esquerdo é uma chave, sendo condição suficiente para obedecer tanto à Forma Normal Boyce-Codd, como à 3a Forma Normal, menos restritiva que a primeira.

Restrições

Na criação da base de dados, foram incluídas as restrições mais convenientes para a manutenção da integridade dos dados armazenados. De um modo não exaustivo, segue-se a enunciação e explicação das várias restrições implementadas.

Restrições Chave (**PRIMARY KEY** ou **UNIQUE**):

O uso deste tipo de restrições serve o facto de nenhum dos atributos associados poder ser repetido, sendo, ou uma chave primária da relação em causa, ou um atributo também identificador dos elementos da tabela, mas que não figure como chave da mesma.

Atributos como *id*, ou *sku*, são comumente tornados chave, justamente pelo facto de, não só, serem únicos, como serem identificadores dos tuplos das suas tabelas, permitindo a necessária eficiência em todas as operações relacionadas.

Exemplos do anteposto são:

- A tabela **Marca**, cuja chave primária é representada pelo *id*, contém também um atributo *nome* UNIQUE. Não haverá duas marcas com o mesmo nome.
- A tabela **Produto**, cuja chave primária é o *sku*, contém também um conjunto de atributos UNIQUE, sendo eles o *nome* e *marca*. Dois produtos não podem ter o mesmo nome e pertencerem à mesma marca.
- A tabela **Cliente**, cuja chave primária é o *id* (inteiro), tem os seus atributos *username* e *email* marcados como UNIQUE, uma vez que não haverá dois clientes com um mesmo nome de utilizador, nem com o mesmo email.
- A tabela **Localizacao**, com o *id* a sua chave primária, e o seu *nome*, UNIQUE.
- A tabela **Estatuto**, com o *id* como chave primária e *título* como UNIQUE.
- Por fim, a tabela **Compra**, cuja chave primária é o *id*, tem no conjunto de atributos *idCliente* e *dataHora* uma chave UNIQUE. Não poderão existir duas compras efetuadas pelo mesmo cliente, à mesma hora.

Restrições de integridade referencial (chaves estrangeiras):

Neste esquema, as chaves estrangeiras são atributos (ou conjuntos de atributos) que referenciam uma chave primária de outra relação. Dado que muitas tabelas contêm chaves estrangeiras, serão enunciados, apenas, os tópicos genéricos e alguns exemplos.

Aquando da conversão do modelo conceptual para o modelo relacional, a forma mais plausível de representar associações *many-to-many* foi criar tabelas auxiliares com chaves estrangeiras para as relações envolvidas, incluindo potenciais atributos presentes em classes de associação.

São exemplo as relações **VcompativelComC**, que materializa o conceito de compatibilidade entre **Videojogos** e **Consolas** (uma consola pode ser compatível com vários videojogos e um videojogo tem de ser compatível com pelo menos uma consola); e **Avaliacao**, que representa as avaliações dos **Clientes** sobre os **Videojogos** (um cliente pode avaliar criticamente vários videojogos e estes podem ter várias avaliações de diferentes clientes).

Associações *many-to-one* viram criada uma chave estrangeira no lado *many*, como referência para a tabela no lado *one*. É exemplo a relação **Produto**, que contém referências para **Marca** e **DescontoEspecial**, já que um produto pode, apenas, ter uma marca e, eventualmente, ter, ou não, um desconto associado.

As generalizações relacionadas com **Produto** e **Conquista** garantiram os ajustamentos enunciados na contextualização anterior e, por esses motivos, as suas classes derivadas ganharam uma chave estrangeira (que passou a ser, também, a chave primária), necessária, nas relações correspondentes. Esta chave estrangeira, *sku* nas tabelas **Acessorio**, **Consola**, **Videojogo**, **Merchandising** e *id* nas tabelas **Fidelizacao**, **NivelPontos**, **NumeroCompras**, referencia as relações **Produto** e **Conquista**, respetivamente.

Todas as outras implementações de restrições de integridade referencial presentes na base de dados são exemplos destes casos aqui enunciados.

Ações Referenciais:

As únicas combinações de ações referenciais em restrições com chaves estrangeiras presentes no nosso esquema são:

1. ON DELETE CASCADE ON UPDATE CASCADE
2. ON DELETE RESTRICT ON UPDATE CASCADE
3. ON DELETE SET NULL ON UPDATE CASCADE

A primeira combinação garante que todas as ações efetuadas na chave pai são propagadas para as chaves filho, nas relações que delas dependem. É usada, sobretudo, em relações que resultaram de associações *many-to-many*, descritas na secção anterior, permitindo que qualquer ação efetuada numa relação correspondente a um extremo de uma dessas associações se reflita imediatamente na tabela que a referencia. Exemplo é a relação **Avaliacao**, onde existem chaves estrangeiras para Videojogo e Cliente, o que significa que apagando, ou modificando, a chave primária de qualquer tuplo comum, a alteração propagar-se-á no tuplo presente na tabela **Avaliação** - a justificação casual para estes casos é que, por exemplo, apagando ou alterando um Videojogo, ou um Cliente, não faz sentido existirem avaliações desatualizadas. O mesmo acontece em todas as outras ocorrências desta combinação de ações.

A segunda impede eliminações de tuplos referenciados, permitindo a propagação de alterações. É útil nas relações que contém informação mais sensível e que são usadas em vários outros campos da base de dados. Assim, proíbem-se as eliminações antes de garantir que informação importante não é perdida. Um exemplo é a tabela **CompraDoProduto** que impede qualquer eliminação de tuplos referenciados presentes tanto na relação Compra, como na relação Produto, evitando perder a informação relativa à compra de uma determinada quantidade um dado produto, por algum cliente.

A última combinação de ações é usada, apenas, em chaves que referenciam Descontos de algum tipo. A sua eliminação levará ao aparecimento da notação NULL no campo correspondente, na tabela que o referencia, voltando ao estado normal de não aplicação de um dado desconto, por exemplo, num **Produto**, como se sucede com o seu atributo **idDesconto**.

Restrições CHECK (a nível dos atributos):

Aqui, existia a limitação de a restrição ser apenas aplicada aos valores dos atributos de uma mesma relação. Todas as outras restrições, mais fortes, mais abrangentes, a nível da base de dados, serão implementadas nas próximas tarefas. Referindo, por exemplo, o facto de o valor de uma compra ser a soma dos preços dos produtos e/ou bundles associados, juntamente com os seus descontos; ou, o rating de um videojogo ser a média das avaliações feitas pelos clientes; ou, ainda, o número de pontos de um cliente ser o total ganho em conquistas e em compras; ou, finalmente, por limitações impostas pelo sistema SQLite, a idade, calculada através da data de nascimento do cliente, ser superior a 13 anos - todos estes exemplos requerem mecanismos mais sofisticados que restrições CHECK.

Excluindo estes casos, as restrições CHECK, como garantia de segurança e controlo na introdução e atualização dos dados, foram implementadas, resumidamente, da seguinte maneira:

- Atributos do tipo 'Mínimo' (sufixo *Min*), ou 'Início', serão menores que atributos do tipo 'Máximo' (sufixo *Max*), ou 'Fim', respetivamente correspondentes. Estes, quando aplicável, e atributos relacionados com quantidade, stock, preço, percentagem (com um máximo de 70%), valor, pontos, etc., serão sempre maiores que zero (ou igual, nos casos pertinentes, como os pontos iniciais de um cliente recém criado, ou, o stock de um produto que esgotou, entre outros...).
- Datas verão o seu formato restringido e unicamente aceite se escritas da forma '%Y-%m-%d'. O mesmo se aplica para horas com formato '%H:%M:%S'.
- O *rating* de um videojogo poderá ser NULL, não tendo, ainda, avaliações, ou ser, tal como o valor de uma avaliação, um número inteiro entre 1 e 5.
- Uma compra pode ter, ou um desconto sazonal, ou um desconto por pontos, ou nenhum desconto associado diretamente, logo, terá de cumprir a restrição *CHECK (idSazonal IS NULL OR idPorPontos IS NULL)*, por forma a não ter dois descontos associados em simultâneo.
- Os *pontos* obtidos por um cliente numa dada compra são a parte inteira do valor da compra e, portanto, assume-se a restrição *CHECK (pontos = CAST(valor AS INTEGER))*.

Restrições NOT NULL:

Ignorando o implícito de que chaves primárias não podem ter valor NULL, este tipo de restrição foi empregue em todos os casos em que, manifestamente, um atributo tem de ter um valor explícito, para garantir a mínima plausibilidade da instância que por si é composta. Posto isto, as explicações mais pertinentes residirão nos casos em que não foi utilizada esta restrição.

- Uma compra pode ter os seus atributos *idSazonal* e *idPorPontos* com valor NULL, já que pode não ter a si associado nenhum desconto, indo de encontro à restrição mencionada na secção anterior.
- Uma conquista não precisa de ter, necessariamente, uma descrição, nem um tipo, sendo, por exemplo, uma conquista genérica, simples.
- Um bundle também não precisa de uma descrição obrigatória.
- Um videojogo, tal como referido anteriormente, pode não ter, ainda, avaliações (*rating* NULL).
- Acessórios, consolas, videojogos, merchandising, têm vários dos seus atributos característicos também opcionais.
- Um produto pode não ter um desconto associado, logo, o seu atributo *idDesconto*, pode ter valor NULL.

Todos os outros atributos são acompanhados desta restrição. Convém, de igual modo, referir o uso de valores DEFAULT, em alguns casos, por uma questão de conveniência.

Interrogações

Segue-se um conjunto de interrogações consideradas pertinentes para o contexto da base de dados, comprovando o correto funcionamento dos vários componentes e suas ligações. As interrogações, aqui descritas em linguagem natural, encontram-se nos respetivos ficheiros, ordenadas pela mesma numeração.

- 1. Apresenta os videojogos lançados em 2019, ordenados por ordem crescente da sua data de lançamento e, no caso de empate, por ordem alfabética do nome do jogo.*
- 2. Apresenta os bundles constituídos por uma consola e um acessório, ordenados por ordem decrescente de preço.*
- 3. Mostra as 3 primeiras Consolas com maior número de Acessorios Compatíveis e, no caso de empate, ordenadas pelo valor descendente de preço.*
- 4. A soma dos pontos ganhos por cada cliente em Conquistas do tipo NivelPontos e NumeroCompras.*
- 5. Número de Conquistas por Localização.*
- 6. Compras com um Desconto Sazonal ou PorPontos onde foi adquirido pelo menos um produto com Desconto Especial.*
- 7. Pares de Clientes que fizeram exatamente a mesma avaliação ao mesmo Videojogo.*
- 8. Produtos mais vezes comprados, incluindo em Bundles.*
- 9. Compras em que foram adquiridos apenas Bundles ou todos os Produtos comprados tinham Desconto.*
- 10. Pares de Produtos/Bundles mais vezes comprados juntos.*

Gatilhos

Havendo necessidade de garantir uma correta monitorização e manutenção da base de dados e controlo sobre as operações realizadas, foram definidos três grupos de gatilhos. Estes encontram-se implementados nos ficheiros respetivos, ordenados pela numeração correspondente.

1. *Maior13Anos*

A inserção de um novo Cliente na base de dados só pode ser aprovada caso tenha uma idade válida, isto é, superior ou igual a 13 anos. As datas são inseridas com o formato “AAAA-MM-DD” e verificadas através da função *JULIANDAY*, que calcula se a data de nascimento do novo cliente é válida, pelo número de dias, com $4745 = 13 \text{ (anos)} * 365 \text{ (dias por ano)}$, abortando a operação em caso de falha.

2. *AtualizarStockProduto, AtualizarStockBundle*

Assim que é realizada e inserida na base de dados uma nova compra, especificando os produtos, ou bundles, comprados, criando tuplos para as relações **CompraDoProduto**, ou **CompraDoBundle**, respetivamente, o stock dos produtos envolvidos deve ser atualizado, ou seja, deve ser decrementado consoante a quantidade comprada. Em termos de **Produto**, o pensamento é simples. Em termos de **Bundle**, significa reduzir ao stock a quantidade do produto associado ao Bundle, multiplicada pela quantidade desse Bundle comprado.

3. *AtualizarRatingInsert, AtualizarRatingDelete, AtualizarRatingUpdate*

Sucintamente, sempre que se realizam operações na tabela **Avaliacao**, o rating dos videojogos associados deve ser atualizado. Para tal, recalcula-se a média das avaliações existentes para o videojogo e converte-se o valor para *Inteiro*, de forma a satisfazer as restrições do atributo. Quando se alteram avaliações existentes, o pensamento é o mesmo e, quando se apagam, além de se atualizar o rating dos videojogos ainda referenciados, os videojogos sem avaliações ganham automaticamente o valor NULL para o atributo em causa.