SGBD de Salão de Jogos

Projeto de Bases de Dados 2020/2021

P9G9:

- NMEC 97606 - Diogo Monteiro
- NMEC 97880 - Camila Fonseca

Índice

Introdução

Para o projeto final de Bases de Dados escolhemos criar um sistema de gestão de uma base de dados de uma franquia de Salões de Jogos, espaços de entretenimento em que clientes podem jogar diversos jogos em variadas máquinas *arcade* e trocar pontos adquiridos por prémios.

Análise de Requisitos

Para esta base de dados, quisemos criar um modelo de uma *franchise* de Salões de Jogos (ou Arcades) e uma interface que permita a gestão desta mesma.

Assim, esta deve:

- Guardar uma lista de **máquinas de arcade**, caracterizadas por:
    - Número de série
    - Jogo
    - Fabricante
    - Fornecedor
    - Tempo de aluguer
    - Custo de aluguer.

Cada máquina tem apenas um **Jogo** e um **Fabricante,** e o fornecedor pode ser alterado ao longo do tempo. No entanto, isto pode apenas acontecer após o tempo de aluguer expirar (Tal como remover a máquina do sistema.)

- Um **jogo** pode ser jogado em várias máquinas, e é caracterizado por:
    - Código interno
    - Nome
    - Número de jogadores
    - Pontos dados
    - Custo por jogada (em créditos)

Um jogo não tem necessariamente de estar associado a uma máquina, e considera-se que este pode mudar de máquina livremente.

- Registar os **fornecedores** das máquinas, que são caracterizados por:
    - NIF
    - Morada
    - Nome
    - Telefone
    - Email

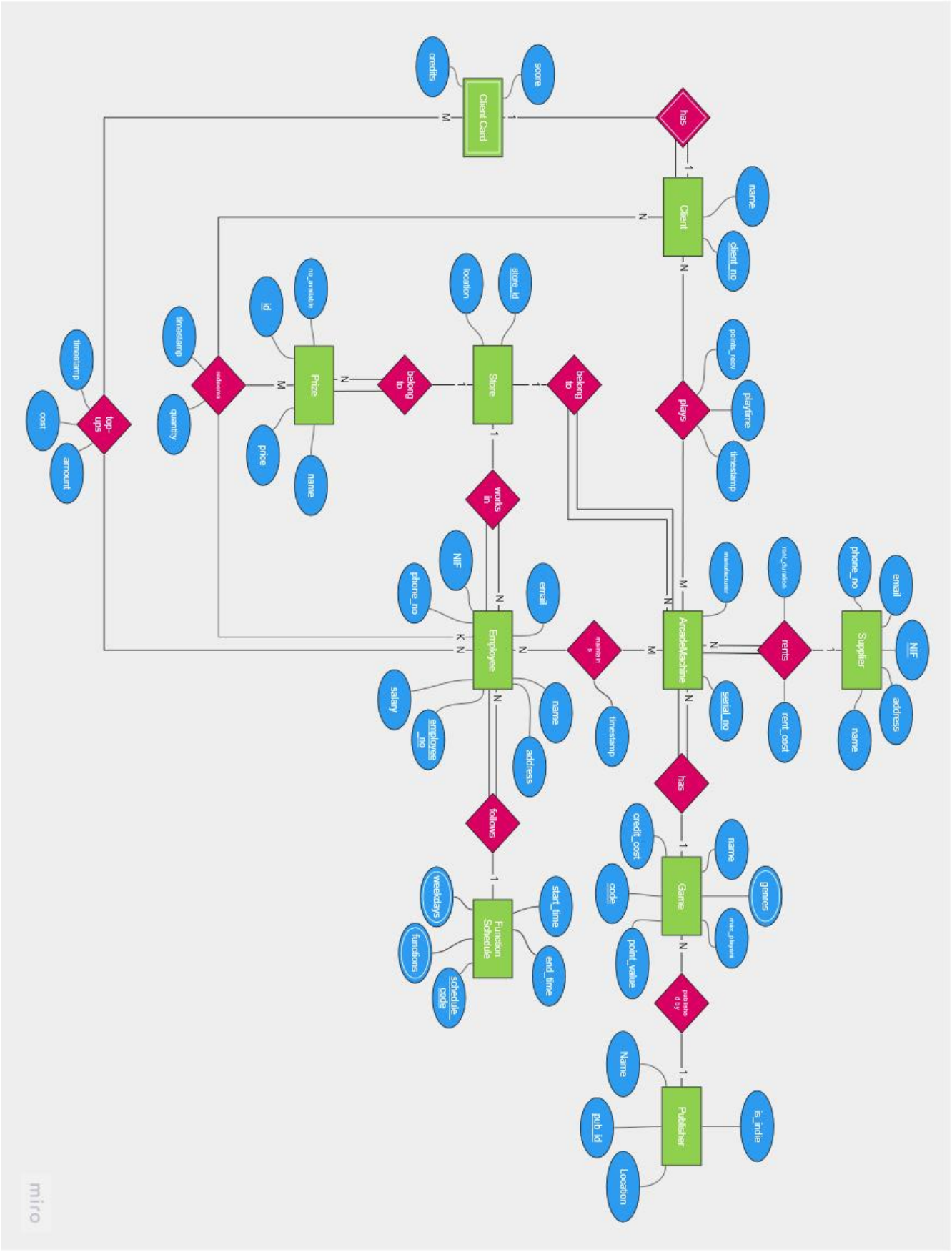
- Guardar os **prémios** que podem ser ganhos, com
    - Nome
    - Número de identificação interno

    - Preço em pontos

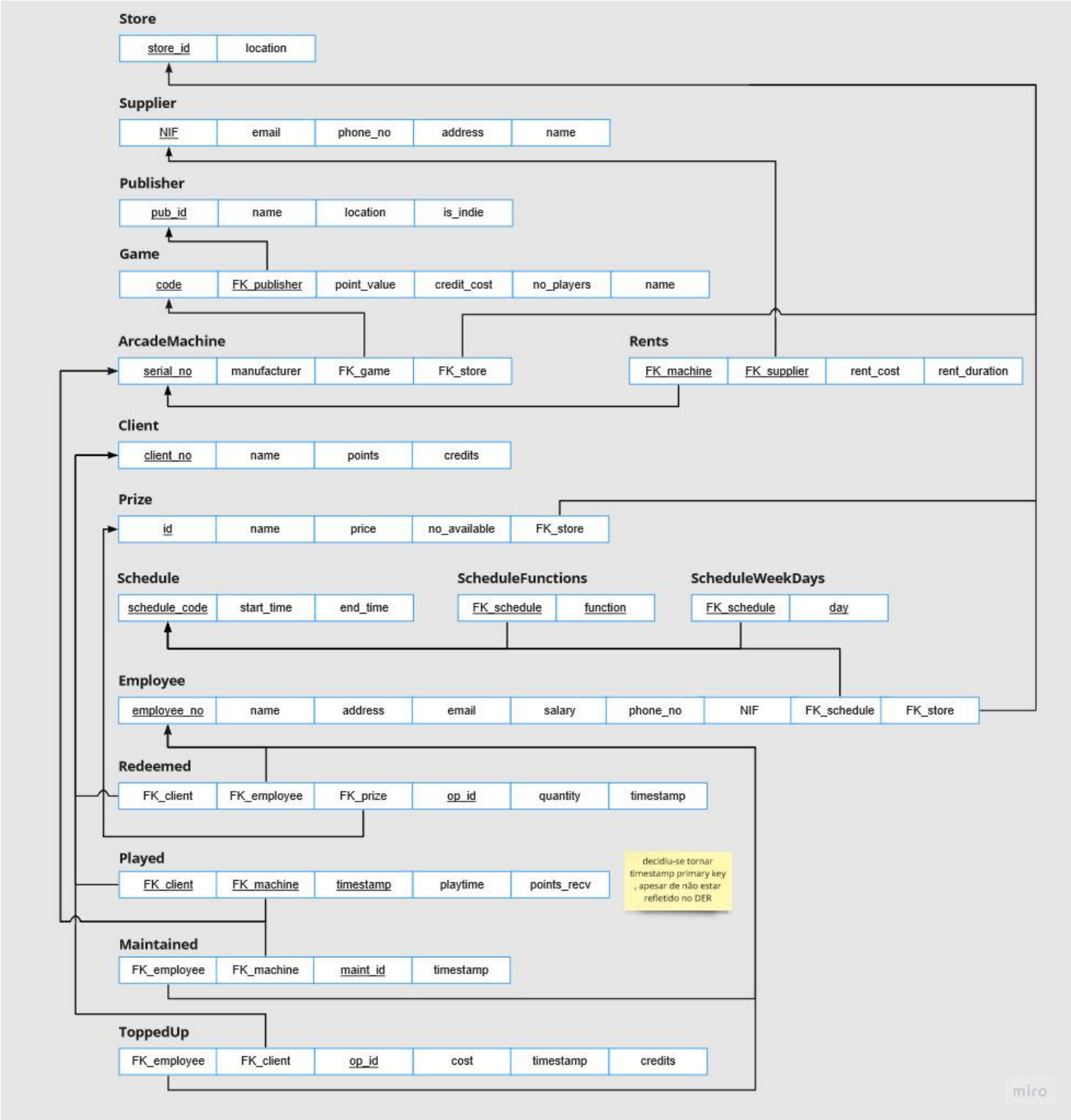
A base de dados deve permitir alterar, inserir ou remover Prémios, Jogos e Máquinas (Juntamente com o seu Fornecedor)

- Registar os **clientes** inscritos na Arcade, que são caracterizados por:
    - Número de cliente
    - Nome
    - Pontos acumulados
    - Balanço de Créditos atuais
- Registar os **empregados** que trabalham na Arcade, que são caracterizados por:
    - Código do número de funcionário
    - Salário
    - Nome
    - Número de cartão de cidadão
    - Morada
    - Data de nascimento
    - Contacto telefónico
    - Email
- Os empregados seguem uma escala de **funções**, que regista as suas funções e horários.
- Quando um cliente joga numa máquina, este ganha pontos, e é guardado um **registo** de jogo com:
    - timestamp
    - máquina usada
    - pontos ganhos
    - duração do jogo
- Os clientes compram tokens de jogo recorrendo a um empregado. É guardada a hora, o custo, e o número de tokens desta operação. É-lhes incrementado o número de tokens disponíveis.
- Os clientes levantam os prémios recorrendo a um empregado. É criado um registo do levantamento, com a hora do levantamento, o prémio e quantidade.

Diagrama de Entidade-Relação



Esquema Relacional



O esquema relacional foi implementado praticamente por inteiro, exceto nas operações relacionadas aos registos (Maintained, Played, ToppedUp e Redeemed), nas quais foi alterada a chave primária para um ID *auto-increment*.

SQL DDL

Para criar a base de dados modelada pelo Esquema Relacional, procedemos à construção de várias queries de criação de tabelas, prestando cuidado à definição de constrições de dados quando apropriadas, como NOT NULL, UNIQUE, formato de strings como email, e valores default. Tivemos também atenção à ordem de criação das tabelas em função das foreign keys, para garantir que uma chave referenciada já existe no momento de criação da tabela.

Depois de criadas as tabelas, fomos ao longo do projeto definindo e criando Índices, Stored Procedures, Triggers e UDFs quando a necessidade surgia.

As queries para a criação de tabelas encontram-se no ficheiro CreateTables.sql

SQL DML

A manipulação da base de dados é feita quase inteiramente com Stored Procedures, para fornecer uma camada de abstração a utilizadores, que não precisam de conhecer o modelo de dados para utilizar o SGBD. Para além disso, esta abstração permite fazer alterações ao modelo de dados sem necessidade de atualizar aplicações que a usem, dado que as SPs também são alteradas para garantir compatibilidade.

Caso sejam feitos INSERTs e DELETEs, a integridade dos dados e certas condições são verificadas com o uso de Triggers, que podem dar ROLLBACK à operação caso estas condições falhem, ou substituir o comportamento normal da operação.

Fizemos com que SPs e Triggers que afetem diversas tabelas ou envolvam várias alterações fossem atómicas com o uso de Transactions, de modo que caso qualquer uma parte do processo falhe, todas as alterações são revertidas. Consideramos que isto seria melhor do que ter os dados apenas parcialmente alterados.

As queries para popular as tabelas com alguns valores exemplo encontram-se no ficheiro DataToInsert.sql

Índices

Cada tabela tem o *clustered index* da *primary key*, criado automaticamente pelo SQL Server. Adicionalmente, criamos alguns *nonclustered indexes* para certas tabelas para que a pesquisa por certos atributos seja mais eficiente.

Segue uma lista dos *nonclustered indexes* que criamos:

Tabela	**Atributos**
ArcadeMachine	- Supplier NIF - Manufacturer
Client	- Name
Employee	- Name - Address - Email - Phone
Game	- Name
Supplier	- Name - Address - Email - Phone Number
Publisher	- Name - Address

O código para criar os índices encontra-se no ficheiro indexes.sql

Triggers

Para diversas tabelas criamos Triggers para a operação de INSERT e/ou DELETE. Segue uma lista dos Triggers criados e os seus efeitos:

Tabela	Operação	Efeito
Supplier	INSTEAD OF DELETE	Verifica se a data limite de aluguer de todas as máquinas do supplier já foi ultrapassada. Caso alguma máquina ainda esteja no período de aluguer, a operação é cancelada - senão, o supplier e todas as suas máquinas são apagadas.
Employee	INSTEAD OF DELETE	Atualiza todas as referências para o *employee* em outras tabelas para o ID de "deleted employee" antes de apagar o *employee*.
Client	INSTEAD OF DELETE	Atualiza todas as referências para o *client* em outras tabelas para o ID de "deleted client" antes de apagar o *client*.
ArcadeMachine	INSTEAD OF DELETE	Atualiza todas as referências para a máquina em outras tabelas para o ID de "deleted machine" antes de apagar a máquina.
Game	AFTER DELETE	Verifica se o publisher do jogo apagado tem algum outro jogo associado. Se nenhum outro jogo existir, o publisher também é apagado.
Schedule	INSTEAD OF DELETE	Verifica se algum employee tem o schedule associado. Caso sim, a operação é cancelada. Senão, apaga o *schedule* assim como os *ScheduleFunctions* e *ScheduleWeekDays* associados.
Redeemed	AFTER INSERT	Verifica se o cliente tem pontos suficientes para dar *redeem* à quantidade inserida, e se existe stock suficiente do prémio. Se sim, atualiza os pontos do cliente e o stock do prémio. Senão, ROLLBACK do INSERT.
Played	AFTER INSERT	Verifica se o cliente tem créditos suficientes para jogar na máquina. Se sim, atualiza os créditos e pontos do cliente. Senão, ROLLBACK do INSERT

O código para criar os Triggers encontra-se no ficheiro Triggers.sql

Stored Procedures

Criamos vários Stored Procedures para intermediar a manipulação dos dados e obter diferentes conjuntos de valores em função dos parâmetros de entrada. É feito um *try-catch* quando é apropriado para que informação detalhada do erro não chegue ao utilizador, optando por apresentar mensagens customizadas.

Segue uma lista dos SPs criados e a sua função:

Stored Procedure	**Parâmetros**	**Função**
playtime_stats	client: INT (opcional), game: INT (opcional), start_date: DATE (opcional), end_date: DATE (opcional), OUTPUTS: avg_time: CHAR(9), total_time: CHAR(10)	Calcula o tempo total e o tempo médio de jogo e retorna os resultados em strings no formato “hh:mm:ss”. Parâmetros de entrada opcionais são usados para filtrar os registos que entram no cálculo.
supplier_machines	supplier: CHAR(9)	Devolve o conjunto de máquinas alugadas pelo *supplier* dado.
game_machines	game: INT	Devolve o conjunto de máquinas que disponibilizam o jogo dado.
publisher_games	pub: INT	Devolve o conjunto de jogos do publisher dado.
topUp	value: INT, client: INT, employee: INT	Incrementa os créditos do *client* por value, e cria um registo da operação.
saleHistory	emp: INT	Devolve o conjunto de operações (redeemed,maintained, toppedUp) em que o *employee* dado este envolvido.
getChanges	user: INT	Devolve movimentos de créditos do cliente dado.
extendRent	machine_id: INT, days: INT (opcional), months: INT (opcional), years: INT (opcional)	Extende a data de fim de aluguer da máquina dada pelo tempo dado.
redeemPrize	client: INT,	Utilizar os pontos de *client* para

	prize: INT, employee: INT, quantity: INT (default 1)	dar *redeem* à quantidade dada de *prize*, com intervenção de *employee*.
addGameToMachine	machine: INT, game: INT	À máquina selecionada via o seu serial_no, altera o jogo que lhe pertence para o passado pelo argumento *game.*
removeGameFromMachine	machine: INT	Remove o jogo da máquina dada, ficando esta sem jogo atribuído.

Para além dos Procedures detalhados na tabela acima, criamos vários SPs para a leitura, inserção, alteração e remoção de entradas nas tabelas. Para brevidade, omitimos a lista completa destes Procedures.

Os Procedures de inserção recebem tipicamente como parâmetros os atributos do tuplo a inserir, à exceção dos registos cujo *timestamp* é obtido automaticamente como sendo a data e hora no momento da inserção.

Procedures de alteração recebem como parâmetro obrigatório a *primary key* da entrada a alterar, e qualquer número de parâmetros de atributos da entrada que podem ser alterados.

Procedures de deleção requerem apenas a *primary key* da entrada a apagar. É efectuada a verificação de se algo pode ser apagado via os *triggers*.

O código para criar os Stored Procedures encontra-se no ficheiro SPs.sql

User Defined Functions

Foram criadas duas simples UDFs para simplificar o acesso aos dados e não fazer uma query grande (e mais lenta) quando apenas precisaríamos de um campo da tabela.

getPrizePrice	ret: INT (Return Value) prize: INT	Devolve o preço de um prémio, através do seu código (PK)
getPrizeStock	ret: INT (Return Value) prize: INT	Devolve o número de prémios de um tipo específico disponíveis, através do seu código (PK)