FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Departamento de Informática
Licenciatura em Engenharia Informática
Base de Dados

Torneios de Cartas Magic

Relatório Final

Grupo 5
Catarina Albino, Nº 31794, P9
Luís Silva, Nº 31698, P9
Rúben Cachucho, Nº 31780, P9

Dezembro de 2010

Índice

1. Tema e objectivos da Base de Dados

O tema da nossa base de dados é “Torneios de Cartas Magic”.

Magic: the Gathering, ou simplesmente **Magic**, é um jogo de cartas coleccionáveis criado por Richard Garfield, no qual cada jogador utiliza um baralho de cartas construído por si (*deck*), para tentar vencer o adversário.

Com esta base de dados pretendemos armazenar informação sobre torneios de Cartas Magic. Pretendemos focar-nos em dois âmbitos principais, que se interligam entre si: informação sobre as **cartas** existentes e informação sobre os **torneios** de Magic.

Uma carta Magic pode ser de vários grupos de cartas, mas por motivos de simplificação, na nossa base de dados vamos só especificar os dois mais abrangentes: uma carta pode ser ou uma *multicard* ou uma *“carta normal”*. Estes dois grupos têm algumas características em comum tais como o tipo de carta, o seu identificador, a raridade, a(s) cor(es), a edição a que pertence e o facto de ser ou não uma carta brilhante.

Uma *multicard* é, no fundo, uma agregação de duas *“cartas normais”* distintas numa única carta (ver anexo B). Estas duas cartas podem ser de cores diferentes, têm habilidades e nomes diferentes, mas partilham o mesmo tipo e raridade. Existem três níveis de raridade - comum, incomum, e rara - e existem seis tipos de cartas - criaturas, instantâneas, encantamentos, feitiços, encantar criatura e terrenos.

Uma *“carta normal”*, para além dos atributos comuns referidos anteriormente, tem um nome, um custo para ser colocada em campo, ou seja, um valor que é necessário “pagar” para a jogar, o custo de mana, que é a representação em termos de cor do custo para entrar em campo e um valor de ataque e um valor de defesa (no caso de a carta ser do tipo criatura). Tem também um texto descritivo onde se encontram informações relevantes sobre as habilidades da carta e um outro texto, designado por *flavour*, que é uma frase caracterizadora do estilo/imagem da carta.

O custo de mana representa os tipos de terrenos que um jogador tem de “pagar” para colocar a carta em campo. Imaginemos que uma carta necessita de dois terrenos de qualquer cor mais um preto e um branco. Então o custo de mana irá ter o valor “2BW”, em que B é *black* e W é *white*. Para além disso, este campo pode guardar, por exemplo, “XBB” em que X é uma representação especial que significa que o jogador pode pagar os terrenos que quiser (sendo os mesmos de qualquer cor) para jogar a carta e utilizar uma qualquer habilidade da carta com os terrenos gastos. No entanto, em termos de custo para entrar em campo, esta carta tem o valor 2.

Os caracteres permitidos para representar os terrenos são:

B – um terreno preto;
W – um terreno branco;
U – um terreno azul;
R – um terreno vermelho;
G – um terreno verde;
Valores numéricos também são permitidos.

Uma *multicard*, por representar duas *"cartas normais"*, tem todos os atributos delas, excepto o poder de ataque, poder de defesa, subtipo e *flavour*. Como estão representadas duas cartas, os seus atributos são a duplicar.

Cada carta, tem de pertencer obrigatoriamente a uma edição, ter pelo menos uma cor e tem de ter um pintor. Uma *multicard* pode ser pintada por mais do que um pintor, num máximo de dois. As edições são caracterizadas por um identificador, nome, ano de lançamento e abreviatura. Existem seis cores de cartas - verde, vermelho, preto, branco, azul e artefacto (incolor) - e cada carta pode ter um número qualquer de cores, excepto se for artefacto, onde não poderá ser de mais nenhuma cor.

Relativamente aos pintores das cartas iremos guardar só o seu nome e nacionalidade.

A base de dados vai guardar informação sobre os *decks* dos vencedores. Cada carta pode pertencer a mais do que um *deck*, podendo existir cartas que não pertencem a nenhum. Um *deck* contém as cartas que o formam e uma descrição do *deck*.

Dos campeões vamos guardar o seu nome, nacionalidade, telefone e data de nascimento.

Cada torneio organizado pode ter várias categorias, ou seja, num mesmo torneio decorrem vários torneios de diferentes categorias, ao mesmo tempo. Pretendemos representar torneios ocorridos em diferentes países e como tal, sobre os locais vamos guardar a cidade, o país, e o recinto onde ocorreu o torneio.

Existem seis tipos de categorias: *standard*, *extended*, *block*, *two-headed giant*, *vintage* e *legacy*.

Um campeão pode vencer várias categorias num mesmo torneio, com o mesmo *deck* ou com um *deck* diferente. Pode também ser vencedor de um mesmo torneio em diferentes datas, ou seja, em diferentes edições desse torneio.

Existem cartas que estão banidas de determinados torneios. As cartas têm uma espécie de nível de restrição (*tear*), que determina em que torneios podem ser jogadas. Uma carta de um determinado nível pode ser usada em torneios desse nível e dos níveis inferiores. Pode considerar-se que o nível *extended* é o nível 1, o *standard* o nível 2 e o *block constructed* o nível 3. Assim, uma carta de nível *standard* pode participar em torneios de nível *standard* e *extended* mas não pode participar no nível *block constructed*. Cada categoria e cada carta têm obrigatoriamente um tear.

Com o objectivo de simplificar a nossa base de dados, não pretendemos guardar informação sobre todos os participantes dos torneios, apenas sobre os vencedores e não guardamos informação sobre todos os formatos de cartas e seus tipos/características e atributos. Caso o fizéssemos a base de dados tornar-se-ia demasiado complexa devido há grande variedade de tipos de cartas com características distintas.

2. Modelo de dados

2.1 Alterações adoptadas em relação à versão inicial

No nosso DER inicial não conseguíamos guardar informação relativa ao *deck* com que um campeão jogou num determinado torneio. Só conseguíamos guardar informação sobre que *deck* o campeão usou numa determinada data, não sendo possível saber em que categoria ganhou com esse *deck*, no caso de ter ganho duas categorias nesse torneio.

Para conseguirmos modelar essa informação tornamos o conjunto de entidades *Decks* num conjunto de entidades forte de *Winners*. Sendo assim *Winners* é um conjunto de entidades fraco, tendo dois conjuntos de entidades fortes. Um tuplo de *Winners* com o atributo *date1* e *idChampion* não era suficiente para se identificar como único, mas com a adição do atributo *idDeck* tal já acontece.

No código da criação da Base de Dados, em várias chaves externas adicionámos a linha de código *"on cascade delete"* para garantir que, no momento em que o tuplo cuja chave "pai" é apagada, todos os tuplos nesta relação que tinham aquela chave "pai" como chave externa serão apagados.

Adicionámos também várias restrições de unicidade para garantir que não existe mais do que um tuplo com o(s) mesmo(s) valor(es) num(em) determinado(s) atributo(s).

Nota: colocamos o nome do atributo de Winners como *date1* porque dava erro se fosse só *date*.

2.2 Discussão das opções tomadas

Como entregue no registo do tema do trabalho, o nosso tema inicial era "Venda de Cartas Magic". Porém, após a criação do modelo de entidades e relações, verificámos em conjunto com o nosso professor das práticas que o modelo era demasiado "circular", isto é, tinha uma tabela principal no centro e uma grande quantidade de tabelas ligadas a esta com relações de 1:N. Apesar de incluirmos uma parte referente às vendas, não tínhamos nem agregações, nem hierarquias, nem entidades fracas e como tal, as consultas que poderíamos efectuar a esta base de dados iriam se revelar demasiado simples.

Depois de discutirmos este ponto com o professor, decidimos adicionar torneios e seus vencedores, retirando a parte das vendas, alterando assim o objectivo da base de dados, que passou a modelar cartas Magic, torneios, vencedores e os *decks* utilizados pelos vencedores dos torneios.

Para representarmos as cartas optámos por criar uma entidade mais geral (*Cards*) contendo os atributos comuns aos dois grupos de cartas existentes (*idCard*, *type*, *foil* e *rarity*) e usar a especialização para designar os dois subgrupos, distintos entre si (*disjoint*), que correspondem a *MultiCards* e *NormalCards*, cada um com os seus respectivos atributos e relações. *Foil* irá conter um valor booleano que identifica a carta como brilhante ou não; *type* e *rarity* irão ser do tipo *varchar* e representarão o tipo de carta e raridade, respectivamente.

A especialização é total, pois uma carta ou é uma *multicard* ou uma *normalcard*. Como uma *MultiCard* pode ser pintada por dois pintores distintos, e uma carta normal apenas por um único pintor, dividimos esta informação em dois conjuntos de relações, com restrições de mapeamento N:M (no caso da relação que interliga *MultiCards* e *Painters*) e N:1 (na que faz a correspondência entre *NormalCards* e *Painters*). A participação dos conjuntos de entidades *Multicards* e *NormalCards* nesses dois conjuntos de relações é, em ambos os casos, total.

Para guardamos a informação relativa às *Multicards*, decidimos criar oito atributos, para guardar o nome, o custo de mana, o texto e o custo para entrar em campo de cada carta. Decidimos guardar a informação desta forma para facilitar posteriores consultas e para tornar a informação mais legível.

No que diz respeito à informação sobre *NormalCards*, utilizámos os atributos nome, custo de mana, subtipo, texto, defesa, ataque, *flavour* e custo para entrar em campo.

Quanto à edição decidimos representá-la como um conjunto de entidades e não como atributo de *Cards*.

No que diz respeito à cor ou cores da carta, optámos igualmente por representá-la através de um conjunto de entidades e não como atributo de *Cards*. A razão principal foi o facto de uma carta poder ter mais que uma cor, e sendo assim, caso a cor fosse um atributo, tal não poderia ser representado.

O conjunto de entidades *Cards* tem participação total nas relações com cor, edição e tear, pois estes são características obrigatórias das cartas.

Criámos um conjunto de entidades para representar *decks* de cartas, com um identificador, um nome e uma descrição. As cartas podem pertencer a um ou vários *decks* diferentes e para além disso, um *deck* pode já ter sido criado e não ter ainda

nenhuma carta associada. Por estas razões o conjunto de relações que associa *Cards* a *decks* (*IsIncluded*) tem restrições de mapeamento N:M e ambas as participações são parciais. Este conjunto de relações tem um atributo *quantidade* que representa a quantidade daquela carta no respectivo *deck*.

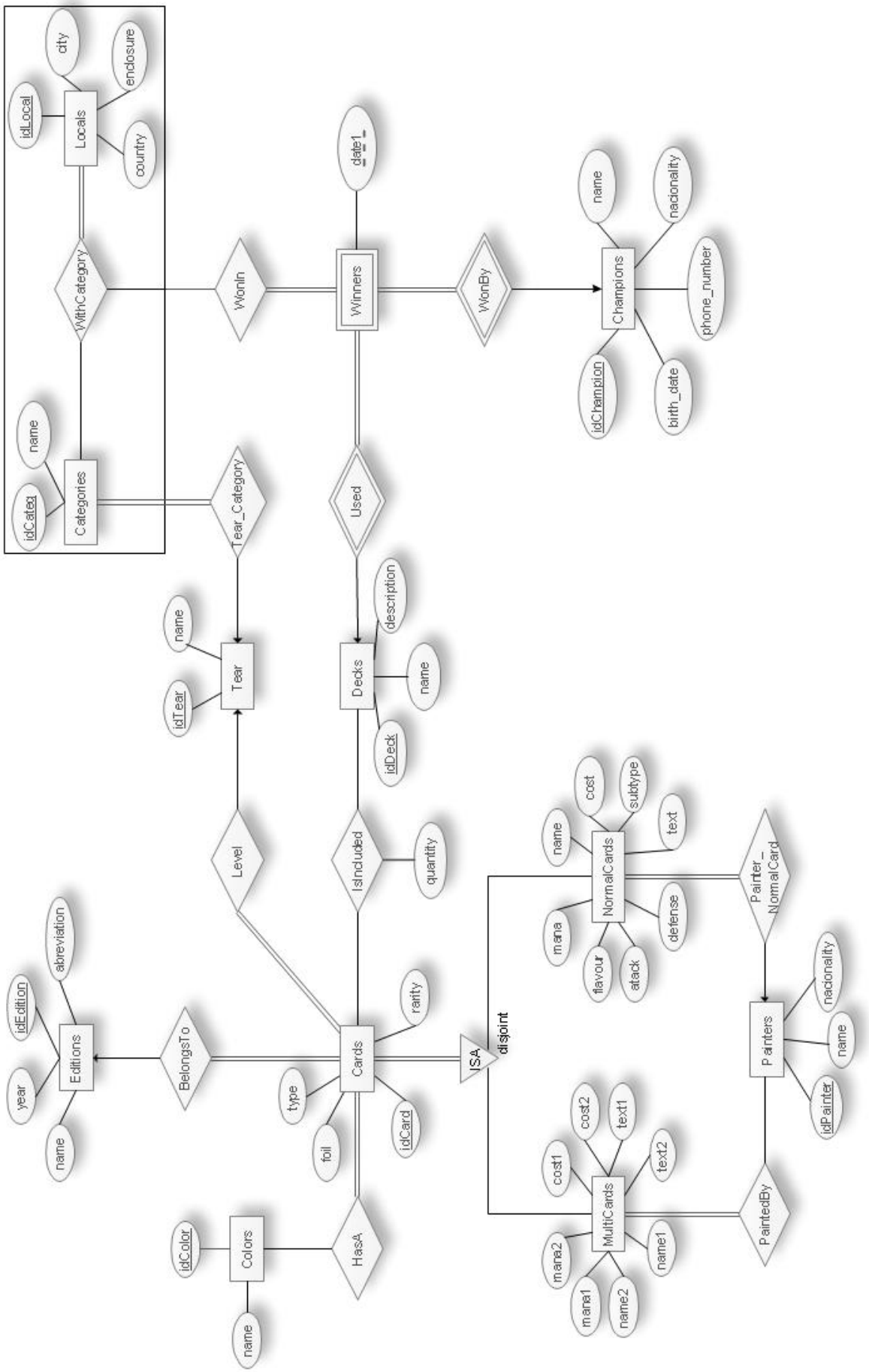
Optámos por criar uma agregação entre locais e categorias, que se relaciona com campeão, pois cada campeão ganha um torneio num certo recinto numa categoria. Colocamos também o conjunto de entidades *Winners* como entidade fraca do conjunto de entidades *Champions*, pois um campeão é vencedor de um torneio numa certa data, mas pode ganhar o mesmo torneio em datas diferentes, ou numa mesma data, um mesmo torneio mas em categorias diferentes. Essa informação fica guardada no conjunto de relações *WonIn*, que relaciona o local do torneio e a categoria com a data do torneio e com o vencedor desse torneio. Em *Winners* fica a informação do *deck* que foi utilizado numa data por um vencedor.

Como existem cartas banidas em alguns torneios, criámos um conjunto de entidades *Tear* que representa o nível de restrição, e relacionámo-lo com o conjunto de entidades *Categories* e com o conjunto de entidades *Cards*. No conjunto de relações que interliga *Cards* a *Tear*, as participações são, respectivamente, total e parcial e a restrição de mapeamento é N:1. No conjunto de relações que interliga *Tear* a *Categories*, as participações são, respectivamente, parcial e total e a restrição de mapeamento é 1:N.

A figura abaixo esquematiza os atributos correspondentes a *Cards*, *NormalCards* e *Multicards*:

	Cards	*NormalCards*	*MultiCards*
idCard	✓	✓	✓
rarity	✓	✓	✓
type	✓	✓	✓
foil	✓	✓	✓
idEdition	✓	✓	✓
idTear	✓	✓	✓
name		✓	
name1			✓
name2			✓
mana		✓	
mana1			✓
mana2			✓
cost		✓	
cost1			✓
cost2			✓
flavour		✓	
attack		✓	
defense		✓	
subtype		✓	
text		✓	
text1			✓
text2			✓
idPainter		✓	

2.3 Modelo ER


idLocal
city
enclosure
country
Locals
WithCategory
idCateg
name
Categories
WonIn
date1_
Winners
WonBy
name
nacionality
Champions
phone_number
idChampion
birth_date
Tear_Category
Used
idTear
name
Tear
description
name
idDeck
Decks
Level
IsIncluded
quantity
idEdition
abreviation
year
name
Editions
BelongsTo
Cards
rarity
type
foil
idCard
ISA
disjoint
name
cost
subtype
text
mana
flavour
attack
defense
NormalCards
Painter_NormalCard
Painters
nacionality
name
idPainter
PaintedBy
cost1
cost2
text1
text2
mana2
mana1
name2
name1
MultiCards
idColor
name
Colors
HasA

2.4 Passagem Modelo ER para o Modelo Relacional

Versão 1

Cards ({ idCard, rarity, type, foil })

Multicards ({ idCard, name1, name2, mana1, mana2, cost1, cost2, text1, text2 })

NormalCards ({ idCard, name, mana, cost, flavour, attack, defense, subtype, text })

Editions ({ idEdition, name, year, abbreviation })

Colors ({ idColor, name })

Tear ({ idTear, name })

Decks ({ idDeck, name, description })

Winners ({ date, idChampion, idDeck })

Champions ({ idChampion, name, nacionality, birth_date, phone_number })

Categories ({ idCateg, name })

Locals ({ idLocal, city, country, enclosure })

Painters ({ idPainter, name, nacionality })

Painter_NormalCard ({ idCard, idPainter })

PaintedBy ({ idCard, idPainter })

BelongsTo ({ idCard, idEdition })

HasA ({ idCard, idColor })

Level ({ idCard, idTear })

Tear_Category ({ idCateg, idTear })

IsIncluded ({ idCard, idDeck, quantity })

WonBy ({ date1, idChampion })

Used ({ date1, idChampion, idDeck })

WithCategory ({ idCateg, idLocal })

WonIn ({ date1, IdChampion, idCateg, idLocal, idDeck })

Versão 2

Nesta versão eliminámos as tabelas redundantes usando chaves-externas (encontram-se a negrito):

Cards ({ idCard, rarity, type, foil, **idEdition**, **idTear** })

Multicards ({ idCard, name1, name2, mana1, mana2, cost1, cost2, text1, text2 })

NormalCards ({ idCard, name, mana, cost, flavour, attack, defense, subtype, text, **idPainter** })

Editions ({ idEdition, name, year, abbreviation })

Colors ({ idColor, name })

Tear ({ idTear, name })

Decks ({ idDeck, name, description })

Winners ({ date1, **idChampion**, **idDeck** })

Champions ({ idChampion, name, nacionality, birth_date, phone_number })

Categories ({ idCateg, name, **idTear** })

Locals ({ idLocal, city, country, enclosure })

Painters ({ idPainter, name, nacionality })

PaintedBy ({ idCard, idPainter })

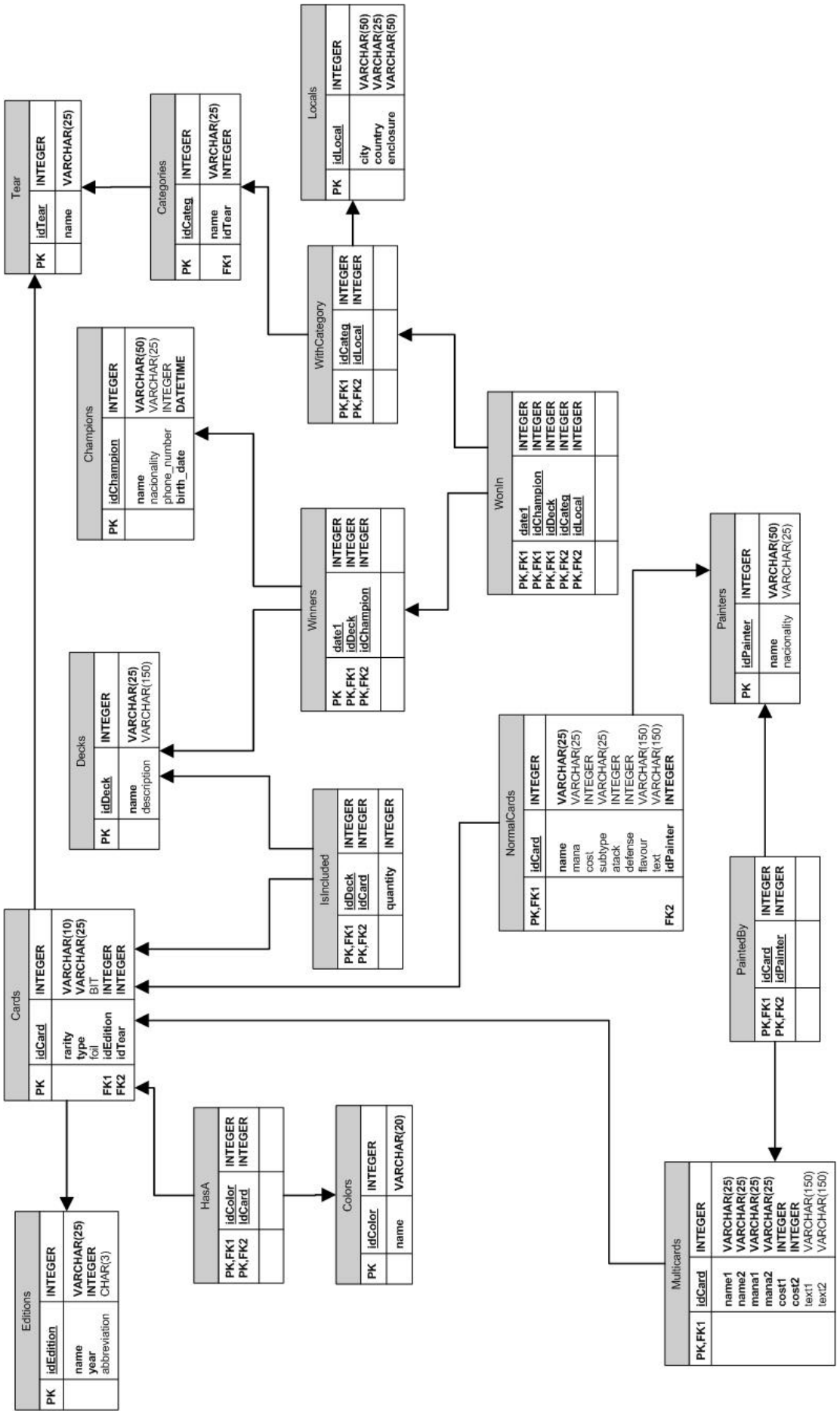
HasA ({ idCard, idColor })

IsIncluded ({ idCard, idDeck, quantity })

WithCategory ({ idCateg, idLocal })

WonIn ({date1, IdChampion, idCateg, idLocal, idDeck })

2.5 Modelo Relacional



2.6 Limitações/opções tomadas para a implementação da BD

Para cada uma das tabelas representadas na implementação da Base de Dados foi necessário criar algumas restrições para garantir o bom funcionamento e integridade da nossa Base de Dados.

Na tabela *Editions* declarámos os atributos *idEdition*, *name*, e *year* como *not null*, pois são campos obrigatórios na definição de uma edição de cartas magic. É garantido que o ano da edição é sempre positivo e que o nome da edição é sempre único, tal como a abreviatura que lhe está associada. A chave primária é *idEdition*.

Na tabela *Colors* todos os seus atributos *(idColor* e *name*) estão declarados como not null, pois são campos obrigatórios na definição de uma cor. Nesta tabela é garantido que o domínio do atributo *name* (nome da cor) é apenas: Black, Red, Green, Blue e Artifact. A chave primária é *idColor*.

Na tabela *Tear* todos os seus atributos (*idTear* e *name*) estão declarados como not null, pois são campos obrigatórios na definição de um nível de restrição de torneio. Nesta tabela é garantido que o domínio do atributo *name* (nome do nível) é apenas: Extended, Standard e Block Constructed. A chave primária é *idTear*.

Na tabela *Decks* os atributos *idDeck* e *name* são declarados not null, pois são campos obrigatórios na definição de um baralho de cartas magic. É garantido que não existe mais do que um deck com o mesmo nome, através de uma restrição de unicidade neste campo. A chave primária é *idDeck*.

Na tabela *Categories* todos os seus atributos (*idCateg, name* e *idTear*) são declarados como not null pois são campos obrigatórios na definição de uma categoria de torneio. É garantido que o domínio do atributo *name* (nome da categoria) é: Standard, Extended, Block, Two-headed Giant, Vintage e Legacy. O atributo *idTear* é chave externa e idCateg é chave primária.

Na tabela *Locals* todos os seus atributos (*idLocal, city, country* e *enclosure*) estão declarados como not null, pois são obrigatórios na definição do local de um torneio. É garantido que para um local só existe uma cidade e um país, através de uma restrição de unicidade. A chave primária é *idLocal.*

Na tabela *Champions* os atributos *idChampion, name*, e *birth_date* estão declarados como not null pois são campos obrigatórios na definição de um campeão. É garantido que o número de telefone de um campeão tem pelo menos 9 dígitos. A chave primária é *idChampion*.

Optámos por criar a restrição de que não existem campeões com idade inferior a 12 anos, usando para tal a data de nascimento.

Na tabela *Cards* os atributos idCard, rarity, type, idEdition e idTear estão declarados como not null pois são campos que definem qualquer carta magic que exista. É ainda garantido que a raridade de uma carta ou é comum, incomum ou rara, tal como é garantido que o tipo de uma carta ou é instantânea, encantamento, criatura, encantar criatura ou terreno. A chave primária é idCard.

Optámos por restringir a remoção das cartas, pois queremos guardar um histórico das cartas usadas pelos vencedores, pelo que só é possível remover cartas que não estejam inseridas em um deck. Relembramos que a nossa BD é referente a torneios de cartas, e se uma carta não está em nenhum deck, tal significa que não foi usada em torneios.

Na tabela *MultiCards* todos os seus atributos são declarados como not null à excepção dos atributos text1 e text2, pois nem todas as cartas contêm um texto descritivo. Os atributos name1 e name2 são declarados como únicos, pois não podem haver cartas com os mesmos nomes. A chave primária é idCard.

Na tabela *NormalCards* os atributos idCard, name e idPainter são declarados como not null, pois são campos que definem qualquer carta normal. Os atributos não declarados desta forma são: *mana* pois, por exemplo, os terrenos não pagam mana para serem jogados; *cost*, pelos mesmos motivos que o atributo *mana*, uma vez que o *cost* designa o total de mana a pagar; *text* pois podem haver cartas sem habilidades; *attack* e *defense* pois as cartas que são criaturas não têm ataque nem defesa; *subtype* pois nem todas as cartas têm um subtipo; e *flavour* pois nem todas as cartas têm um texto descritivo. A chave primária é idCard.

É ainda garantido que o custo de uma carta, se existir, nunca é negativo, e o mesmo é garantido para o ataque e defesa de uma carta. O nome de uma carta é declarado como único pois não podem haver cartas diferentes como o mesmo nome.

Na tabela *Winners* todos os seus atributos (date1, idDeck e idChampion) são declarados not null, pois um campeão tem que ter sempre associados uma data de vitória e o deck usado. Todos os atributos são chave primária.

A tabela *PaintedBy* todos os seus atributos (idCard e idPainter) estão declarados como not null, pois cada multicard (correspondente ao atributo idCard) tem que ter um pintor. Ambos os atributos são chave primária da tabela.

A tabela *HasA* contem os atributos idCard e idColor que são chaves externas, pois para carta é necessário associar uma cor, portanto os atributos são declarados como not null. Ambos os atributos são chave primária.

A tabela *IsIncluded* contém os atributos idCard, idDeck e quantity declarados como not null, pois cada carta está inserida num deck numa determinada quantidade que nunca é superior a 4. A chave primária é idCard e idDeck, que também são chaves externas.

A tabela *WithCategory* contém os atributos idLocal, idCateg que são chaves externas, pois a cada local onde ocorre um torneio estão atribuídas categorias. Ambos os atributos são chave primária.

A tabela *WonIn* contem os atributos date1, idLocal, idCateg, idChampion e idDeck pois a cada vencedor é necessário atribuir qual o local onde foi efectuado o torneio que ganhou e qual a categoria, tal como o deck usado e a data em que venceu. Desta forma. Todos os atributos são chaves externas e são chave primária.

Criámos também vistas para efectuarmos consultas, nomeadamente:

- *All_Multi_Cards* e *All_Normal_Cards* que permitem visualizar toda a informação referente a *multicards* e a *normalcards*.
- V_WonIn que mostra todos os dados da tabela *WonIn*. Vai servir para garantir a integridade desta tabela.
- V_PaintedBy que mostra todos os dados da tabela *PaintedBy*. Vai servir para garantir a integridade desta tabela.
- V_HasA que mostra todos os dados da tabela HasA. Vai servir para garantir a integridade desta tabela.

Desta forma é fornecido um nível de abstracção relativamente ao modelo lógico da base de dados aos utilizadores, garantindo a segurança dos dados.

Se a Base de Dados fosse implementada com finalidade de ser utilizada no meio empresarial, os vários utilizadores teriam diferentes níveis de restrições de acesso à Base de Dados, mas nenhum iriam ter direito a efectuar alterações directamente nas tabelas *Cards*, *MultiCards*, *NormalCards*, *HasA*, *PaintedBy* e *WonIn*. A inserção de dados efectua-se, por isso, sobre as vistas criadas.

Na Base de Dados optámos por guardar apenas a informação referente aos vencedores dos torneios e não de todos os participantes pois o objectivo é apenas guardar a informação dos vencedores para ter acesso aos *decks* por estes usados e, consequentemente as cartas usadas, pelo que não demonstramos interesse em guardar informação referente a todos os participantes.

2.7 Criação da Base de Dados

2.7.1 Criação das Tabelas

```
drop table Editions cascade constraints;

create table Editions( idEdition number(10) not null,
          name varchar(25) not null,
          year number(4) not null check(year >= 0),
          abbreviation char(3),
          CONSTRAINT edition_Abbrev UNIQUE (abbreviation),
          CONSTRAINT edition_NameAbbrev UNIQUE (name, abbreviation),
          primary key( idEdition) );


drop table Colors cascade constraints;

create table Colors( idColor number(10) not null,
          name varchar(20) not null
          check(name in('Black', 'White', 'Red', 'Green', 'Blue',
'Artefact')),
          primary key(idColor) );


drop table Tear cascade constraints;

create table Tear( idTear number(10) not null,
          name varchar(25) not null
          check (name in ('Extended', 'Standard', 'Block Constructed')),
          primary key(idTear) );


drop table Decks cascade constraints;

create table Decks( idDeck number(10) not null,
          name varchar(25) not null,
          description varchar(150),
          CONSTRAINT deck_Name UNIQUE (name),
          primary key(idDeck) );


drop table Categories cascade constraints;
```

```
create table Categories( idCateg number(10) not null,
          name varchar(25) not null
          check (name in('Standard', 'Extended', 'Block', 'Two-headed
Giant', 'Vintage', 'Legacy')),
          idTear number(10) not null,
          primary key(idCateg),
          foreign key(idTear) references Tear ON DELETE CASCADE);


drop table Locals cascade constraints;

create table Locals( idLocal number(10) not null,
           city varchar(50) not null,
           country varchar(25) not null,
           enclosure varchar(50) not null,
           CONSTRAINT uniq_local UNIQUE (city, country, enclosure),
           primary key(idLocal) );



drop table Painters cascade constraints;

create table Painters( idPainter number(10) not null,
           name varchar(50) not null,
           nacionality varchar(25),
           primary key(idPainter) );



drop table Champions cascade constraints;

create table Champions( idChampion number(10) not null,
          name varchar(50) not null,
          nacionality varchar(25),
          birth_date date not null,
          phone_number number(15)
          check( LENGTH(phone_number)>= 9),
          primary key(idChampion) );




drop table Cards cascade constraints;

create table Cards( idCard number(10) not null,
           rarity varchar(10) not null check( rarity in('Common',
'Uncommon', 'Rare') ),
           type varchar(25) not null,
           check( type in('Instant', 'Sorcery', 'Enchantment',
'Creature', 'Enchant Creature', 'Land' ) ),
           foil number(1) check( foil = 0 or foil = 1),
           idEdition number(10) not null,
           idTear number(10) not null,
           primary key(idCard),
           foreign key(idEdition) references Editions ON DELETE SET
NULL,
           foreign key(idTear) references Tear ON DELETE SET NULL);
```

```
drop table Multicards cascade constraints;

create table Multicards(idCard number(10) not null,
          name1 varchar(25) not null,
           name2 varchar(25) not null,
           mana1 varchar(25) not null,
           mana2 varchar(25) not null,
           cost1 number(3) not null check( cost1 >= 0),
           cost2 number(3) not null check(cost2 >= 0),
           text1 varchar(150),
           text2 varchar(150),
           CONSTRAINT uniq_names UNIQUE (name1,name2),
           primary key(idCard),
           foreign key(idCard) references Cards ON DELETE CASCADE);



drop table NormalCards cascade constraints;

create table NormalCards(
           idCard number(10) not null,
           name varchar(25) not null,
           mana varchar(25),
           cost number(10) check(cost >= 0),
           flavour varchar(150),
           attack number(3) check (attack >=0),
           defense number(3) check (defense >=0),
           subtype varchar(25),
           text varchar(150),
           idPainter number(10) not null,
           CONSTRAINT uniq_name UNIQUE (name),
           primary key(idCard),
           foreign key(idCard) references Cards ON DELETE CASCADE,
           foreign key(idPainter) references Painters ON DELETE
CASCADE);



drop table Winners cascade constraints;

create table Winners(date1 date not null,
            idDeck number(10) not null,
            idChampion number(10) not null,
            primary key(date1, idChampion, idDeck),
            foreign key(idChampion) references Champions ON DELETE
CASCADE,
            foreign key(idDeck) references Decks ON DELETE CASCADE);



drop table PaintedBy cascade constraints;

create table PaintedBy( idCard number(10) not null,
             idPainter number(10) not null,
             primary key(idCard, idPainter),
             foreign key(idCard) references Cards ON DELETE CASCADE,
             foreign key(idPainter) references Painters ON DELETE
CASCADE);
```

```
drop table HasA cascade constraints;

create table HasA(  idCard number(10) not null,
             idColor number(10) not null,
             primary key (idCard, idColor),
             foreign key(idCard) references Cards ON DELETE CASCADE,
             foreign key(idColor) references Colors ON DELETE CASCADE);


drop table IsIncluded cascade constraints;

create table IsIncluded( idCard number(10) not null,
             idDeck number(10) not null,
             quantity number(1) not null check (quantity > 0 and
quantity <= 4),
             primary key(idCard, idDeck),
             foreign key(idCard) references Cards ON DELETE CASCADE,
             foreign key(idDeck) references Decks ON DELETE CASCADE);


drop table WithCategory cascade constraints;

create table WithCategory( idLocal number(10) not null,
              idCateg number(10) not null,
              primary key( idCateg, idLocal),
              foreign key(idCateg) references Categories ON DELETE
CASCADE,
              foreign key(idLocal) references Locals ON DELETE CASCADE);



drop table WonIn cascade constraints;

create table WonIn( date1 date not null,
              idLocal number(10) not null,
              idCateg number(10) not null,
              idChampion number(10) not null,
              idDeck number(10) not null,
              primary key(date1, idChampion, idCateg, idLocal, idDeck),
              foreign key(date1, idChampion, idDeck) references Winners
                    ON DELETE CASCADE,
              foreign key(idCateg, idLocal) references WithCategory
                    ON DELETE CASCADE);
```

2.7.2 Criação de Sequências

Criámos sequências para as tabelas *Cards*, *Editions*, *Colors*, *Tear*, *Decks*, *Champions*, *Categories*, *Locals* e *Painters*:

```
drop sequence seq_Cards;
create sequence seq_Cards increment by 1 start with 1;


drop sequence seq_Editions;
create sequence seq_Editions increment by 1 start with 1;


drop sequence seq_Colors;
create sequence seq_Colors increment by 1 start with 1;


drop sequence seq_Tear;
create sequence seq_Tear increment by 1 start with 1;


drop sequence seq_Decks;
create sequence seq_Decks increment by 1 start with 1;


drop sequence seq_Champions;
create sequence seq_Champions increment by 1 start with 1;


drop sequence seq_Categories;
create sequence seq_Categories increment by 1 start with 1;


drop sequence seq_Locals;
create sequence seq_Locals increment by 1 start with 1;


drop sequence seq_Painters;
create sequence seq_Painters increment by 1 start with 1;
```

2.7.3 Criação de Vistas

```
CREATE OR REPLACE VIEW  "ALL_MULTI_CARDS" ("IDCARD", "NAME1", "NAME2",
    "MANA1", "MANA2", "COST1", "COST2", "TEXT1", "TEXT2", "RARITY",
"TYPE",
    "FOIL", "IDEDITION", "IDTEAR") AS
select "IDCARD", "NAME1", "NAME2", "MANA1", "MANA2", "COST1", "COST2"
,"TEXT1",
       "TEXT2", "RARITY", "TYPE", "FOIL", "IDEDITION", "IDTEAR"
from MultiCards natural inner join Cards;
/


CREATE OR REPLACE VIEW  "ALL_NORMAL_CARDS" ("IDCARD", "NAME", "TYPE",
    "RARITY", "FOIL", "IDEDITION", "IDTEAR", "MANA", "COST", "FLAVOUR",
    "ATTACK", "DEFENSE", "SUBTYPE", "TEXT", "IDPAINTER") AS
select "IDCARD", "NAME", "TYPE", "RARITY", "FOIL", "IDEDITION",
"IDTEAR",
       "MANA", "COST", "FLAVOUR", "ATTACK", "DEFENSE", "SUBTYPE",
"TEXT",
       "IDPAINTER"
from cards natural inner join NormalCards
order by name;
/


CREATE OR REPLACE VIEW  "V_WONIN" ("PK", "DATE1", "IDCHAMPION",
"IDCATEG",
    "IDLOCAL", "IDDECK") AS
select ROWID pk, DATE1, IDCHAMPION, IDCATEG, IDLOCAL, IDDECK
from WonIn
order by date1;
/


CREATE OR REPLACE VIEW  "V_PAINTEDBY" ("IDCARD", "IDPAINTER") AS
select idCard, idPainter
from PaintedBy;
/


CREATE OR REPLACE VIEW  "V_HASA" ("IDCARD", "IDCOLOR") AS
select idCard, idColor
from HasA;
/
```

2.7.4 Criação de *Triggers*

```
create or replace trigger DEL_ALL_MULTI_CARDS
    INSTEAD OF DELETE on ALL_MULTI_CARDS
    REFERENCING OLD AS OLD
    FOR EACH ROW
DECLARE
    rowcnt number;
begin
      SELECT COUNT(*) INTO rowcnt FROM IsIncluded
          where idCard = :OLD.idCard;
      if rowcnt = 0 then
          delete from Cards
          where idCard = :OLD.idCard;
      else
        RAISE_APPLICATION_ERROR(-20001,'Erro, esta carta pertence a um
deck.');
      end if;
end;
/
```

O *trigger* "DEL_ALL_MULTI_CARDS" garante que só são eliminadas cartas, neste caso, *NormalCards*, que não pertençam a nenhum *deck*, ou seja, cujo identificador idCard não esteja presente na tabela *IsIncluded*. Caso a carta já pertença a algum *deck* a eliminação é interrompida e um erro é apresentado.

```
create or replace trigger DEL_ALL_NORMAL_CARDS
  INSTEAD OF DELETE on ALL_NORMAL_CARDS
  REFERENCING OLD AS OLD
  FOR EACH ROW
DECLARE
    rowcnt number;
begin
      SELECT COUNT(*) INTO rowcnt FROM IsIncluded
          where idCard = :OLD.idCard;
      if rowcnt = 0 then
          delete from Cards
          where idCard = :OLD.idCard;
      else
        RAISE_APPLICATION_ERROR(-20001,'Erro, esta carta pertence a um
deck.');
      end if;
end;
/
```

Este *trigger* tem a mesma finalidade do anterior, com a diferença que, neste caso, as cartas a eliminar são *MultiCards* e não *NormalCards*.

```
create or replace TRIGGER del_wonin
  INSTEAD OF DELETE
   ON v_wonin
  REFERENCING OLD AS OLD
   FOR EACH ROW
BEGIN
 /*DELETE FROM wonin where ROWID = :OLD.pk;*/
 DELETE FROM winners WHERE date1 = :OLD.date1
     and idChampion = :OLD.idChampion and idDeck = :OLD.idDeck;
END;
/
```

Os seguintes oito *triggers* têm como objectivo o preenchimento automático da chave de um tuplo, aquando da sua inserção, recorrendo às sequências anteriormente criadas para cada uma das tabelas *Editions*, *Colors*, *Painters*, *Tear*, *Decks*, *Champions*, *Categories* e *Locals*:

```
create or replace trigger idEditions
before insert on Editions
for each row
declare
    edition_num number;
begin
    select seq_Editions.nextval
    into edition_num
    from dual;
    :new.idEdition := edition_num;
end;
/
```

```
create or replace trigger idColors
before insert on Colors
for each row
declare
    color_num number;
begin
    select seq_Colors.nextval
    into color_num
    from dual;
    :new.idColor := color_num;
end;
/
```

```
create or replace trigger idPainters
before insert on Painters
for each row
declare
    painter_num number;
begin
    select seq_Painters.nextval
    into painter_num
    from dual;
    :new.idPainter := painter_num;
end;
/


create or replace trigger idTear
before insert on Tear
for each row
declare
    tear_num number;
begin
    select seq_Tear.nextval
    into tear_num
    from dual;
    :new.idTear := tear_num;
end;
/


create or replace trigger idDecks
before insert on Decks
for each row
declare
    deck_num number;
begin
    select seq_Decks.nextval
    into deck_num
    from dual;
    :new.idDeck := deck_num;
end;
/


create or replace trigger idChampions
before insert on Champions
for each row
declare
    champion_num number;
begin
    select seq_Champions.nextval
    into champion_num
    from dual;
    :new.idChampion := champion_num;
end;
/
```

```
create or replace trigger idCategories
before insert on Categories
for each row
declare
    category_num number;
begin
    select seq_Categories.nextval
    into category_num
    from dual;
    :new.idCateg := category_num;
end;
/


create or replace trigger idLocals
before insert on Locals
for each row
declare
    local_num number;
begin
    select seq_Locals.nextval
    into local_num
    from dual;
    :new.idLocal := local_num;
end;
/




create or replace trigger maxPainters
INSTEAD OF insert on V_PaintedBy
REFERENCING NEW AS NEW
for each row
declare
    painter_num number;
    multi_card number;
begin

    select count(*) into painter_num from PaintedBy where idCard =
:New.idCard;
    select count(*) into multi_card from MultiCards where idCard =
:New.idCard;

      if multi_card = 0 then
        RAISE_APPLICATION_ERROR(-20001,'Erro, esta carta não é uma
multiCard.');
      end if;

      if painter_num < 2 then
          INSERT INTO PaintedBy
          VALUES (:NEW.idCard, :NEW.idPainter);
      else
       RAISE_APPLICATION_ERROR(-20001,'Erro, esta carta já tem dois
pintores.');
      end if;

end;
/
```

O *trigger* "maxPainters" garante a restrição de que, para uma *MultiCard*, só podem existir no máximo dois pintores distintos.
Para tal, aquando da inserção de um novo tuplo na vista V_*PaintedBy*, verifica-se primeiramente se o identificador da carta a inserir representa uma *MultiCard*. Em caso afirmativo é verificado se o número de pintores para essa carta é menor que dois. Se tal acontecer a inserção do novo tuplo é permitida; caso contrário é interrompida sendo mostrado um erro.

```
create or replace trigger cardColor
  instead of insert on V_HasA
  REFERENCING NEW AS NEW
  for each row
  declare n_colors number;
begin

    select count(*) into n_colors from Colors c inner join HasA h
        on c.idColor = h.idColor
        where c.name like 'Artefact' and :NEW.idCard = h.idCard;

    if n_colors = 1 then
       RAISE_APPLICATION_ERROR(-20001,'Erro, esta carta é um
artefacto.');
    end if;

    select count(*) into n_colors from NormalCards c
        where :NEW.idCard = c.idCard;
    if n_colors = 1 then
     select mana_color_normal(:NEW.idColor,:NEW.idCard) into n_colors
from dual;
    else
     select mana_color_multi (:NEW.idColor,:NEW.idCard) into n_colors
from dual;
    end if;

      insert into HasA
      values (:NEW.idCard, :NEW.idColor);
end;
/
```

Este *trigger* verifica aquando da inserção de um tuplo na *view V_HasA,* duas integridades: garante que uma carta, sendo da cor artefacto, não pode pertencer a mais nenhuma cor e verifica se a carta é da cor que *:NEW.idColor* representa, utilizando para tal as funções *mana_color_normal* e *mana_color_multi* consoante a especialidade da carta.

Nota: após a verificação da especialidade da carta, por distracção esquecemo-nos de fazer o código da verificação da segunda integridade e de levantar o erro.

```
create or replace trigger "INS_ALL_MULTI_CARDS"
    INSTEAD OF INSERT
    ON ALL_MULTI_CARDS
    REFERENCING NEW AS NEW
    FOR EACH ROW
    declare card_num number;
BEGIN

    if confirm_mana(:NEW.mana1) = 1 or confirm_mana(:NEW.mana1) = 1 then
       RAISE_APPLICATION_ERROR(-20001,'Erro, mana incorrecta.');
    end if;

    select seq_Cards.nextval
    into card_num
    from dual;

    INSERT INTO Cards
    VALUES (card_num, :NEW.rarity, :NEW.type, :NEW.foil, :NEW.idEdition,
          :NEW.idTear);

    INSERT INTO MultiCards
    VALUES (card_num, :NEW.name1, :NEW.name2, :NEW.mana1, :NEW.mana2,
        :NEW.cost1, :NEW.cost2, :NEW.text1, :NEW.text2);

END;
/
```

O trigger "INS_ALL_MULTI_CARDS" é responsável por garantir a integridade da base dados aquando da inserção de uma MultiCard na vista “ALL_MULTI_CARDS”.
Recorre-se primeiramente à função “confirm_mana” (descrita na secção de criação de funções) para verificar se os atributos mana1 e mana2 foram especificados correctamente. Se a função retornar o valor um para alguns desses atributos, a inserção da nova carta é interrompida e é mostrado um erro.[1] Caso contrário procede-se à inserção de um novo tuplo na tabela *Cards* (com apenas os atributos idCard, rarity, type, foil, edition e tear) e, posteriormente, um novo tuplo na tabela *MultiCards* contendo idCard e os restantes atributos.

[1] Por lapso, os dois atributos presentes no teste são *mana1*. Um deles deveria ser *mana2*.

```
create or replace trigger "INS_ALL_NORMAL_CARDS"
    INSTEAD OF INSERT
    ON ALL_NORMAL_CARDS
    REFERENCING NEW AS NEW
    FOR EACH ROW
    declare card_num number;
BEGIN

    if confirm_mana(:NEW.mana) = 1 then
       RAISE_APPLICATION_ERROR(-20001,'Erro, mana incorrecta.');
    end if;

    if :NEW.type in('Instant', 'Sorcery', 'Enchantment', 'Enchant
Creature' ) and
        (:NEW.attack is not null or :NEW.defense is not null) then
            RAISE_APPLICATION_ERROR(-20001,'Erro, o tipo de carta '
            || :NEW.type || ' não pode ter ataque e/ou defesa.');
    end if;

    select seq_Cards.nextval
    into card_num
    from dual;

    INSERT INTO Cards
    VALUES (card_num, :NEW.rarity, :NEW.type, :NEW.foil, :NEW.idEdition,
          :NEW.idTear);

    INSERT INTO NormalCards
    VALUES (card_num, :NEW.name, :NEW.mana, :NEW.cost, :NEW.flavour,
        :NEW.attack, :NEW.defense, :NEW.subtype, :NEW.text,
:NEW.idPainter);
END;
/
```

Este trigger é responsável por garantir a integridade da base dados aquando da inserção de uma NormalCard na vista “ALL_NORMAL_CARDS”.

Recorre-se primeiramente à função “confirm_mana” (descrita na secção de criação de funções) para verificar se o atributo *mana* foi especificado correctamente. Se a função retornar o valor um, a inserção da nova carta é interrompida e é mostrado um erro.

Garante-se também que uma carta cujo tipo for 'Instant', 'Sorcery', 'Enchantment' ou 'Enchant Creature' não possui ataque nem defesa. Se tal acontecer não é completada a inserção e um erro é mostrado.

Caso contrário procede-se à inserção de um novo tuplo na tabela *Cards* (com apenas os atributos idCard, rarity, type, foil, edition e tear) e, posteriormente, um novo tuplo na tabela *NormalCards* contendo idCard e os restantes atributos.

```
create or replace TRIGGER ins_wonin
 INSTEAD OF INSERT
 ON v_wonin
 REFERENCING NEW AS NEW
 FOR EACH ROW
DECLARE
    rowcnt number;
BEGIN

    SELECT COUNT(*) INTO rowcnt FROM WithCategory
    where (:NEW.idLocal, :NEW.idCateg) in
                (Select idLocal, idCateg from WithCategory);

    if rowcnt = 0 then
     RAISE_APPLICATION_ERROR(-20001,'Erro, este local nem tem essa
categoria.');
    end if;

    SELECT COUNT(*) INTO rowcnt FROM WonIn
    where (:NEW.idLocal, :NEW.idCateg, :NEW.idChampion, :NEW.date1) in
                (Select idLocal, idCateg, idChampion, date1 from WonIn);

    if rowcnt >0 then
     RAISE_APPLICATION_ERROR(-20001,'Erro, este campeão já ganhou o
torneio
            com outro deck.');
    end if;

    SELECT COUNT(*) INTO rowcnt FROM WonIn
    where (:NEW.idLocal, :NEW.idCateg, :NEW.date1) in
                (Select idLocal, idCateg, date1 from WonIn);

    if rowcnt > 0 then
     RAISE_APPLICATION_ERROR(-20001,'Erro, este torneio já tem um
campeão.');
    end if;

    SELECT COUNT(*) INTO rowcnt FROM IsIncluded i inner join
        Cards c on i.idCard=c.idCard, Categories cat
    where i.idDeck = :NEW.idDeck and cat.idCateg = :NEW.idCateg and
        tear_level(tear_name(c.idTear)) >
tear_level(tear_name(cat.idTear));

    if rowcnt > 0 then
     RAISE_APPLICATION_ERROR(-20001,'Erro, o deck não pode jogar nessa
        categoria.');
    end if;

    INSERT INTO winners
    VALUES (:NEW.date1, :NEW.idDeck, :NEW.idChampion);

    INSERT INTO wonin
 VALUES (:NEW.date1, :NEW.idLocal, :NEW.idCateg, :NEW.idChampion,
:NEW.idDeck);

END;
/
```

Este *trigger* verifica toda a integridade referente à tabela *WonIn* na inserção de um novo tuplo na vista *v_wonin.*

Em primeiro lugar verificamos se categoria em questão existe no local. Para isso verificamos se o tuplo *(:NEW.idLical, :NEW.idCateg)* existe no conjunto de relações *WithCategory*.

De seguida verificamos se o torneio já foi ganho por esse campeão mas utilizando outro *deck,* ou se o torneio em questão já tem um vencedor.

Averiguamos também se o *deck* em questão está em condições de participar no torneio, ou seja, se o nível de restrição do *deck* está de acordo com o da categoria do torneio. Para isso temos de verificar se o tear de todas as cartas pertencentes ao *deck* é menor ou igual ao tear do torneio.

```
create or replace TRIGGER UP_ALL_MULTI_CARDS
 INSTEAD OF UPDATE
 ON ALL_MULTI_CARDS
 REFERENCING NEW AS NEW
 FOR EACH ROW
BEGIN

    if confirm_mana(:NEW.mana1) = 1 or confirm_mana(:NEW.mana1) = 1 then
       RAISE_APPLICATION_ERROR(-20001,'Erro, mana incorrecta.');
    end if;

 UPDATE Cards
   SET rarity = :NEW.rarity,
       type = :NEW.type,
       foil = :NEW.foil,
       idEdition = :NEW.idEdition,
       idTear = :NEW.idTear
 WHERE idCard = :NEW.idCard;

 UPDATE MultiCards
   SET name1 = :NEW.name1,
       name2 = :NEW.name2,
       mana1 = :NEW.mana1,
       mana2 = :NEW.mana2,
       cost1 = :NEW.cost1,
       cost2 = :NEW.cost2,
       text1 = :NEW.text1,
       text2 = :NEW.text2
 WHERE idCard = :NEW.idCard;
END;
/
```

O trigger “UP_ALL_MULTI_CARDS” assegura a integridade aquando da actualização de um tuplo na vista

ALL_MULTI_CARDS.

Primeiramente é verificado, através o uso da função “confirm_mana” se os valores dos atributos *mana1* e *mana2* foram correctamente introduzidos.[2] Se para algum deles isso não acontecer, é mostrado um erro e a operação não é completada.

Caso contrário é actualizado o tuplo correspondente na tabela *Cards* e, posteriormente na tabela *MultiCards*.

```
create or replace TRIGGER UP_ALL_NORMAL_CARDS
 INSTEAD OF UPDATE
 ON ALL_NORMAL_CARDS
 REFERENCING NEW AS NEW
 FOR EACH ROW
BEGIN

    if :NEW.type in('Instant', 'Sorcery', 'Enchantment', 'Enchant
Creature' ) and
        (:NEW.attack is not null or :NEW.defense is not null) then
            RAISE_APPLICATION_ERROR(-20001,'Erro, o tipo de carta '
            || :NEW.type || ' não pode ter ataque e/ou defesa.');
    end if;

    if confirm_mana(:NEW.mana) = 1 then
       RAISE_APPLICATION_ERROR(-20001,'Erro, mana incorrecta.');
    end if;

 UPDATE Cards
   SET rarity = :NEW.rarity,
       type = :NEW.type,
       foil = :NEW.foil,
       idEdition = :NEW.idEdition,
       idTear = :NEW.idTear
 WHERE idCard = :NEW.idCard;

 UPDATE NormalCards
   SET name = :NEW.name,
       mana = :NEW.mana,
       cost = :NEW.cost,
       flavour = :NEW.flavour,
       attack = :NEW.attack,
       defense = :NEW.defense,
       subtype = :NEW.subtype,
       text = :NEW.text,
       idPainter = :NEW.idPainter
 WHERE idCard = :NEW.idCard;
END;
/
```

[2] Por lapso, os dois atributos presentes no teste são *mana1*. Um deles deveria ser *mana2*.

Este trigger assegura a integridade aquando da actualização de um tuplo na vista "ALL_NORMAL_CARDS".

É garantido, à semelhança do que é feito no trigger "INS_ALL_NORMAL_CARDS", que uma carta cujo tipo for 'Instant', 'Sorcery', 'Enchantment' ou 'Enchant Creature' não possui ataque nem defesa. Se tal acontecer não é completada a actualização e um erro é apresentado.

Caso contrário é actualizado o tuplo correspondente na tabela *Cards* e, posteriormente na tabela *NormalCards*.

```
create or replace TRIGGER up_wonin
 INSTEAD OF UPDATE
 ON v_wonin
 REFERENCING NEW AS NEW OLD AS OLD
 FOR EACH ROW
DECLARE
    rowcnt number;
BEGIN

    SELECT COUNT(*) INTO rowcnt FROM WithCategory
    where (:NEW.idLocal, :NEW.idCateg) in
                (Select idLocal, idCateg from WithCategory);

    if rowcnt = 0 then
     RAISE_APPLICATION_ERROR(-20001,'Erro, este local nem tem essa
categoria.');
    end if;

    SELECT COUNT(*) INTO rowcnt FROM WonIn
    where (:NEW.idLocal, :NEW.idCateg, :NEW.idChampion, :NEW.date1) in
                (Select idLocal, idCateg, idChampion, date1 from WonIn);

    if rowcnt >0 then
     RAISE_APPLICATION_ERROR(-20001,'Erro, este campeão já ganhou o
torneio
            com outro deck.');
    end if;

    SELECT COUNT(*) INTO rowcnt FROM WonIn
    where (:NEW.idLocal, :NEW.idCateg, :NEW.date1) in
                (Select idLocal, idCateg, date1 from WonIn);

    if rowcnt > 0 then
     RAISE_APPLICATION_ERROR(-20001,'Erro, este torneio já tem um
campeão.');
    end if;

    SELECT COUNT(*) INTO rowcnt FROM IsIncluded i inner join
        Cards c on i.idCard=c.idCard, Categories cat
    where i.idDeck = :NEW.idDeck and cat.idCateg = :NEW.idCateg and
        tear_level(tear_name(c.idTear)) >
tear_level(tear_name(cat.idTear));
```

```
    if rowcnt > 0 then
     RAISE_APPLICATION_ERROR(-20001,'Erro, o deck não pode jogar nessa
        categoria.');
    end if;

      INSERT INTO winners
      VALUES (:NEW.date1, :NEW.idDeck, :NEW.idChampion);

      UPDATE wonin
        SET date1 = :NEW.date1,
           idLocal = :NEW.idLocal,
           idCateg = :NEW.idCateg,
           idChampion = :NEW.idChampion,
           idDeck = :NEW.idDeck
        WHERE ROWID = :NEW.pk;
/*
     DELETE FROM winners WHERE date1 = :OLD.date1
     and idChampion = :OLD.idChampion and idDeck = :OLD.idDeck;
*/

END;
/
```

O trigger *up_wonin* garante todas as integridades garantidas pelo *trigger ins_wowin* utilizando as mesmas funções.

Inserimos o tuplo com a nova informação da data, *deck* e campeão na tabela *Winners* porque precisamos dessa informação para a inserirmos na tabela *WonIn*. Contudo não podemos alterar nem apagar esse tuplo da tabela *Winners* pois ao fazê-lo podemos colocar a base de dados num estado inconsistente (basta considerar que um mesmo campeão no mesmo torneio ganhou com o mesmo *deck* duas categorias).

```
create or replace trigger edition_year
before insert on Editions
REFERENCING NEW AS NEW
for each row
begin
      if (:NEW.year > (extract (year from SYSDATE))) then
        RAISE_APPLICATION_ERROR(-20001,'Erro: ano inválido.');
      end if;
end;
/
```

O trigger “edition_year” assegura a restrição de que um o ano de uma edição existente nunca pode ser maior que o ano actual. Para tal, aquando da inserção de um novo tuplo na tabela *Editions*, o atributo *year* é comparado com o ano da data actual. Se for maior, a inserção do novo tuplo é interrompida é mostrado um erro.

```
create or replace trigger champion_age
before insert on Champions
REFERENCING NEW AS NEW
for each row
begin
      if  (trunc((sysdate-:NEW.birth_date)/365.25)) < 12 then
        RAISE_APPLICATION_ERROR(-20001,'Erro: idade não permitida.');
      end if;
end;
/
```

O trigger “champion_age” assegura a restrição de que um campeão nunca pode ter menos de doze anos de idade. Para tal, aquando da inserção de um novo tuplo na tabela *Champions*, o atributo *birth_date* subtraído à data actual e é efectuado o cálculo referente à obtenção da idade do campeão. Se a idade for menor que doze anos, a inserção do novo tuplo é interrompida é mostrado um erro.

```
create or replace trigger win_date
before insert on Winners
REFERENCING NEW AS NEW
for each row
begin
      if (:NEW.date1 > sysdate ) then
        RAISE_APPLICATION_ERROR(-20001,'Erro: data inválida.');
      end if;
end;
/
```

O trigger que se segue garante que o campo date1 da tabela *Winners*, que representa a data em que um campeão ganha um torneio, nunca é maior que a data actual. Se tal acontecer a inserção do novo tuplo é interrompida é mostrado um erro.

2.7.5 Criação de Funções

```
create or replace function tear_level(tear IN VARCHAR2) return NUMBER is
begin

  if tear like 'Extended' then
    RETURN 1;
  end if;

  if tear like 'Standard' then
    RETURN 2;
  end if;

  if tear like 'Block Constructed' then
    RETURN 3;
  end if;

  RETURN -1;
end;
/
```

A função *tear_level* garante um nível de abstracção em relação ao nível de restrição das cartas. Como foi dito anteriormente, as cartas têm três níveis de restrição aos quais podem ser atribuídos números: *extended* é o nível 1, *standard* o nível 2 e *block constructed* o nível 3.

O objectivo desta função é, recebendo por parâmetro o nome do *tear* da carta, devolver o valor numérico correspondente para futura comparação com outros *tears.*

A função garante que o valor do *tear Standard* é sempre superior ao valor do *tear Extended*, ou seja, garante a ordem correcta dos *tears*.

```
create or replace function tear_name(tear NUMBER) return VARCHAR2
is
  name_tear VARCHAR2(25);
begin
  select name into name_tear from Tear where idTear = tear;
  RETURN name_tear;
end;
/
```

A função *tear_name* devolve o nome do *tear* cujo identificador é passado por parâmetro.

Criámos esta função com a finalidade de a usar na função *total_cards*, para obtenção do nome do *tear*.[3]

[3] Tentámos primeiramente efectuar um um *inner join* para encontrarmos o nome do *tear* (para posterior uso na função *tear_level)*, mas não estava a funcionar correctamente. Resolvemos o problema criando a função.

```
create or replace function total_cards(tear IN VARCHAR2) return NUMBER
is
    total NUMBER;
begin
    SELECT count(*)
    INTO total
    FROM Cards c
    WHERE tear_level(tear_name(c.idTear)) <= tear_level(tear);

    RETURN total;
end;
/
```

Esta função devolve o numero de cartas com o nível de *tear* menor ou igual ao *tear* passado por parâmetro, isto é, devolve o número de cartas que podem participar num torneiro cujo nível de restrição seja o passado por parâmetro.

```
create or replace function confirm_mana (mana IN VARCHAR2) return NUMBER
is
begin

  if NVL(REGEXP_COUNT( mana, '[BWRGUX0123456789]' ), -1) = LENGTH(mana)
then
    RETURN 0;
  end if;
  RETURN 1;

end;
/
```

A função “confirm_mana” devolve zero se a mana de uma carta passada por parâmetro tiver somente os caracteres esperados. Para tal utilizamos a função REGEXP_COUNT que conta o número de vezes que um padrão aparece dentro de uma string. Usámos os parênteses rectos para devolver a soma do número de ocorrências de cada carácter (especificado dentro dos parênteses) na string.

Para confirmar se a mana está correcta comparamos o resultado retornado pela função REGEXP_COUNT com o número total de caracteres da string: se forem iguais é porque a mana foi especificada correctamente.

Recorremos também à função NVL para evitarmos problemas com valores *null* que possam ser passados por parâmetro.

```
create or replace function mana_color_normal (idColor NUMBER, Card
NUMBER)
return NUMBER is
  rowcnt NUMBER;
begin

  -- verify white color
  select count(*) into rowcnt from NormalCards nc
      where nc.idCard = Card and nc.mana like '%W%';
  if rowcnt = 1 then
    select count(*) into rowcnt from HasA h natural inner join Colors c
      where h.idCard = Card and c.name like 'White';
    if rowcnt = 0 then
      RETURN -1;
    end if;
  end if;

  -- verify gree color
  select count(*) into rowcnt from NormalCards nc
      where nc.idCard = Card and nc.mana like '%G%';
  if rowcnt = 1 then
    select count(*) into rowcnt from HasA h natural inner join Colors c
      where h.idCard = Card and c.name like 'Green';
    if rowcnt = 0 then
      RETURN -1;
    end if;
  end if;

  -- verify red color
  select count(*) into rowcnt from NormalCards nc
      where nc.idCard = Card and nc.mana like '%R%';
  if rowcnt = 1 then
    select count(*) into rowcnt from HasA h natural inner join Colors c
      where h.idCard = Card and c.name like 'Red';
    if rowcnt = 0 then
      RETURN -1;
    end if;
  end if;

  -- verify black color
  select count(*) into rowcnt from NormalCards nc
      where nc.idCard = Card and nc.mana like '%B%';
  if rowcnt = 1 then
    select count(*) into rowcnt from HasA h natural inner join Colors c
      where h.idCard = Card and c.name like 'Black';
    if rowcnt = 0 then
      RETURN -1;
    end if;
  end if;

  -- verify blue color
  select count(*) into rowcnt from NormalCards nc
      where nc.idCard = Card and nc.mana like '%U%';
  if rowcnt = 1 then
    select count(*) into rowcnt from HasA h natural inner join Colors c
      where h.idCard = Card and c.name like 'Blue';
    if rowcnt = 0 then
      RETURN -1;
    end if;
  end if;
```

```
  RETURN 0;

end;
/
```

Esta função verifica se a *NormalCard* (cujo id é passado por parâmetro) tem a cor com o identificador *idColor* e, em caso afirmativo, devolve zero. Por exemplo, uma carta para ser Branca e Preta, o valor da sua *mana* tem que conter os caracteres W e B.

Para verificarmos se a carta é da cor branca, verificamos primeiramente se o carácter W aparece no seu atributo *mana* e se tal acontecer, verificamos se já existe o tuplo composto pelo *id* da carta e pelo *id* da cor no conjunto de relações *HasA*.

A verificação para as restantes cores é análoga, trocando somente os valores das cores.

```
create or replace function mana_color_multi (idColor NUMBER, Card
NUMBER)
return NUMBER is
  rowcnt NUMBER;
begin

  -- verify white color
  select count(*) into rowcnt from MultiCards mc
      where mc.idCard = Card and (mc.mana1 like '%W%' or mc.mana2 like
'%W%');
  if rowcnt = 1 then
    select count(*) into rowcnt from HasA h natural inner join Colors c
      where h.idCard = Card and c.name like 'White';
    if rowcnt = 0 then
      RETURN -1;
    end if;
  end if;

  -- verify gree color
  select count(*) into rowcnt from MultiCards mc
      where mc.idCard = Card and (mc.mana1 like '%G%' or mc.mana2 like
'%G%');
  if rowcnt = 1 then
    select count(*) into rowcnt from HasA h natural inner join Colors c
      where h.idCard = Card and c.name like 'Green';
    if rowcnt = 0 then
      RETURN -1;
    end if;
  end if;

  -- verify red color
  select count(*) into rowcnt from MultiCards mc
      where mc.idCard = Card and (mc.mana1 like '%R%' or mc.mana2 like
'%R%');
  if rowcnt = 1 then
    select count(*) into rowcnt from HasA h natural inner join Colors c
      where h.idCard = Card and c.name like 'Red';
    if rowcnt = 0 then
      RETURN -1;
```

```
    end if;
  end if;

  -- verify black color
  select count(*) into rowcnt from MultiCards mc
      where mc.idCard = Card and (mc.mana1 like '%B%' or mc.mana2 like
'%B%');
  if rowcnt = 1 then
    select count(*) into rowcnt from HasA h natural inner join Colors c
      where h.idCard = Card and c.name like 'Black';
    if rowcnt = 0 then
      RETURN -1;
    end if;
  end if;

  -- verify blue color
  select count(*) into rowcnt from MultiCards mc
      where mc.idCard = Card and (mc.mana1 like '%U%' or mc.mana2 like
'%U%');
  if rowcnt = 1 then
    select count(*) into rowcnt from HasA h natural inner join Colors c
      where h.idCard = Card and c.name like 'Blue';
    if rowcnt = 0 then
      RETURN -1;
    end if;
  end if;

  RETURN 0;

end;
/
```

Esta função verifica se a *MultiCard "*Card" (cujo id é passado por parâmetro) tem a cor com o identificador *idColor* e, em caso afirmativo, devolve zero.

Utiliza a mesma estratégia da função "*mana_color_normal"*, verificando se o carácter referente à cor se encontra nos dois atributos, *mana1* e *mana2*.

2.8 Descrição da Interface

A nossa interface contém uma página inicial que contém um menu onde é possível aceder às páginas subsequentes. Pode aceder-se às páginas *ReportCards, Report Decks, Report Categories, Report of Champions, Normal Cards* e a *Create / Edit Cards*.

Na página *Report Cards* podemos consultar todas as cartas existentes na base de dados. Para distinguir entre *multicards* e *normalcards* colocámos, no caso de se tratar de uma *multicards*, os atributos *name* e *mana* são constituídos por o nome e mana (respectivamente) das duas cartas que compõem, sendo separados por uma barra. Nesta página, para além desses atributos podemos consultar o tipo, a raridade e a edição das cartas.

As colunas do tipo, raridade e edição têm um *link* para a página *Normal Cards*. Se carregarmos num determinado tipo de carta, na página *Normal Cards* irão aparecer apenas cartas desse tipo. O mesmo acontece para a raridade e para a edição.

Na página *Report Decks* podemos ver todas as cartas que pertencem a um determinado *deck*. Para tal basta seleccionar o *deck* em questão numa *select list* que se encontra na parte superior da página. Entre a *select list* e a listagem das cartas é mostrada informação sobre o número total de cartas no *deck* e, deste número, quantas cartas são magias e quantas são criaturas. De entre a informação das cartas podemos ver a quantidade dessa carta neste *deck*.

Se na *select list* não seleccionarmos nenhum *deck* não será mostrada nem a listagem das cartas nem a informação referente às características do deck.

Na página *Report Categories* é listado todas as categorias existentes na base de dados bem como o número de cartas que podem jogar nessa categoria. Se carregarmos numa categoria outra página é aberta onde são mostradas essas cartas. No cimo desta outra página vemos o nome da categoria seleccionada anteriormente e em baixo vemos a informação referente às cartas.

A página *Report of Champions* é um *Master detail* onde no *master* podemos ver os campeões e no *detail* os torneios ganhos pelo respectivo *master*. No *master* podemos ver o nome, nacionalidade, data de nascimento e número de telefone do campeão e no lado superior direito encontra-se o botão de criação de um novo campeão. Na criação de um novo campeão somos obrigados a colocar o nome e a data de nascimento.

Ao carregar no botão que permite editar o campeão somos levados para uma página que permite essa edição, na qual vemos o *detail* com a informação referente aos torneios. No *detail* podemos ver a data, categoria, local e com que *deck* foi ganho o torneio e temos também a hipótese de adicionar e/ou remover vitórias em torneios.

Na página *Normal Cards* são listadas todas as *Normal Cards* da base de dados, com toda a informação relevante sobre as mesmas. Nesta página temos na parte superior um conjunto de *select list* que inicialmente têm o valor predefinido *-Any-*. Nestas *select list* podemos filtrar a listagem de cartas. Existem cinco *select list*: para a edição, tipo, subtipo, raridade e cor.

Se o valor da *select list* estiver a *-Any-* significa que não queremos filtrar esse atributo. Ao seleccionarmos um outro valor, só irão aparecer cartas com essa característica. Podemos seleccionar em várias *select list,* já que a listagem irá ser filtrada para as cartas que tenham todas essas características. Na *select list* da cor a filtragem é diferente: irão aparecer cartas que tenham a cor seleccionada, podendo uma carta aparecer em várias cores desde que seja dessas cores.

Depois de encontrarmos a carta pretendida podemos editá-la carregando na imagem do lápis ao nado do nome. Nesta edição de cartas só é permitido editar os atributos da tabela *Cards* e da tabela *NormalCards*, ou seja, não podemos editar chaves estrangeiras.

Na página *Create / Edit Cards* aparece um *radio group* para escolhermos a especialidade de cartas que queremos editar e/ou criar: *Normal Cards* ou *Multi Cards*. Após a escolha carrega-se no botão *Select* para sermos redirigidos para a respectiva página.

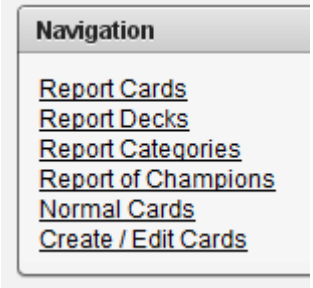
Se escolhermos *Normal Cards* irá aparecer uma página com toda a informação das cartas. Aqui podemos criar novas cartas carregando no botão *Create* e editar todos os atributos (incluído as chaves estrangeiras) de cartas existentes, clicando no lápis de edição. Ao escolhermos *Multi Cards* irá aparecer uma página idêntica à página dos *Normal Cards*, mas com os respectivos atributos.

Em todas as páginas de listagem existe, na parte superior, uma *Search Bar* para pesquisas que se pretendam realizar.

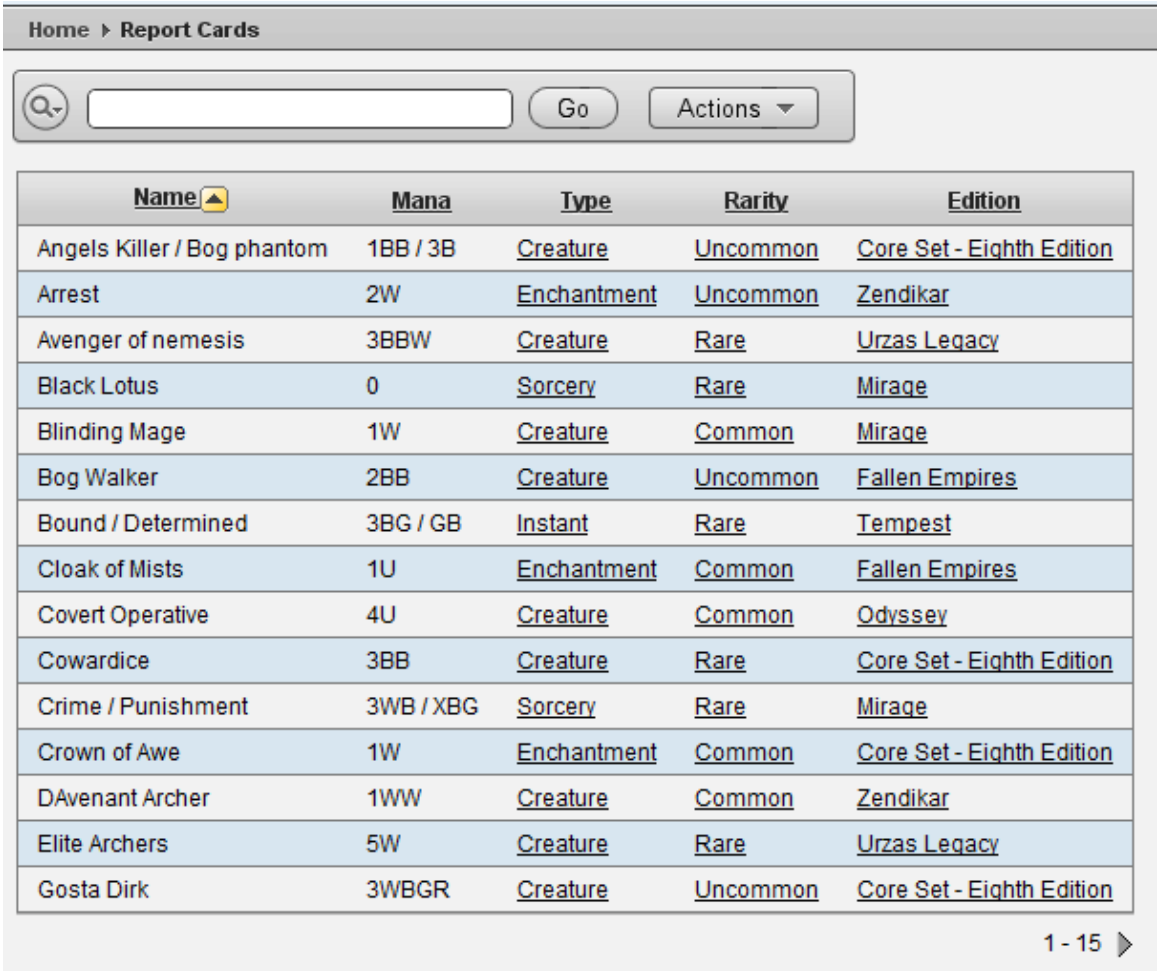
Em todas as páginas onde se procede à edição de valores o *Help Text* tem uma pequena descrição acerca do conteúdo esperado para o campo seleccionado.

3. Manual do utilizador

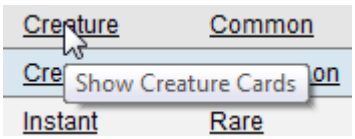
Página inicial com as ligações para as páginas subsequentes.

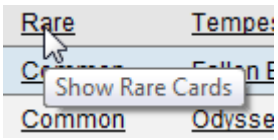


Ao carregarmos no *link Report Cards* aparecerá a seguinte página:

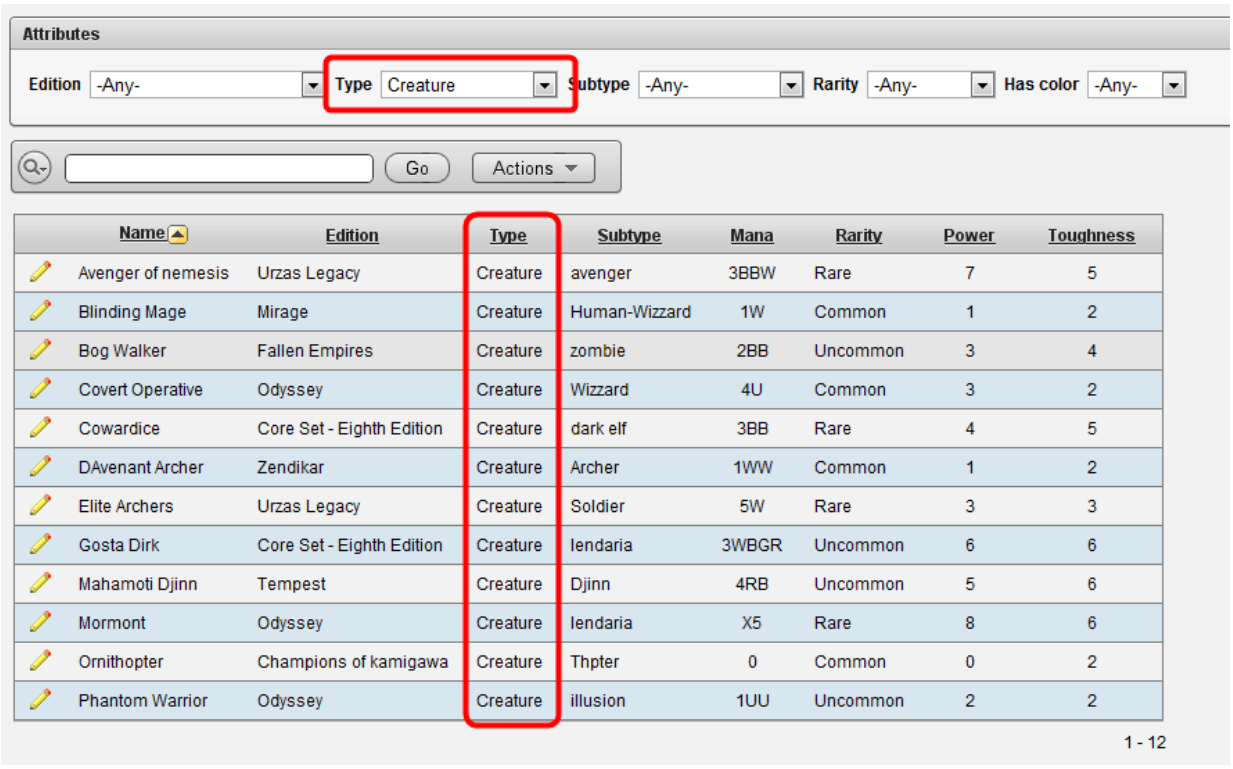


Ao passar por cima dos tipos das cartas, da raridade e da edição aparecerá por baixo do ícone rato uma *tooltip* com uma pequena nota informativa.

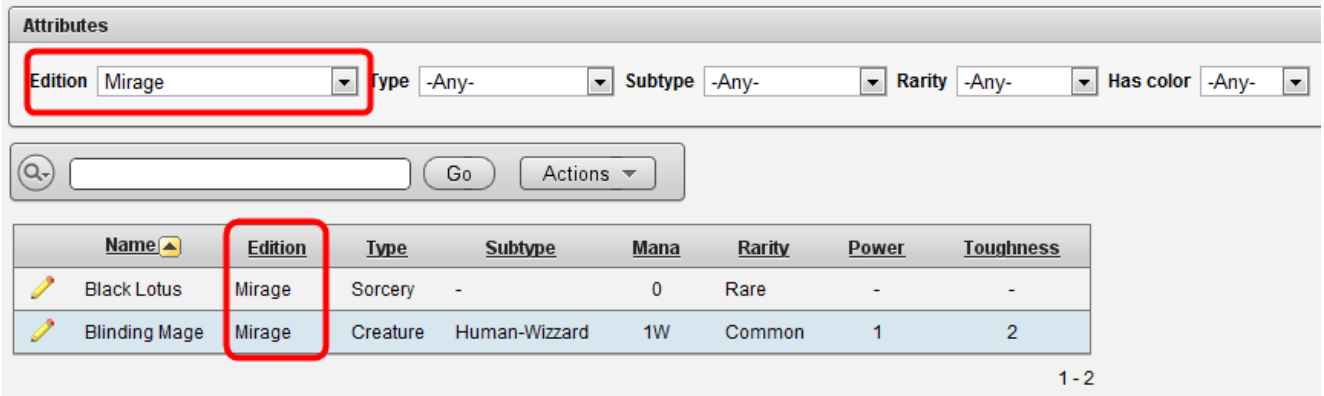




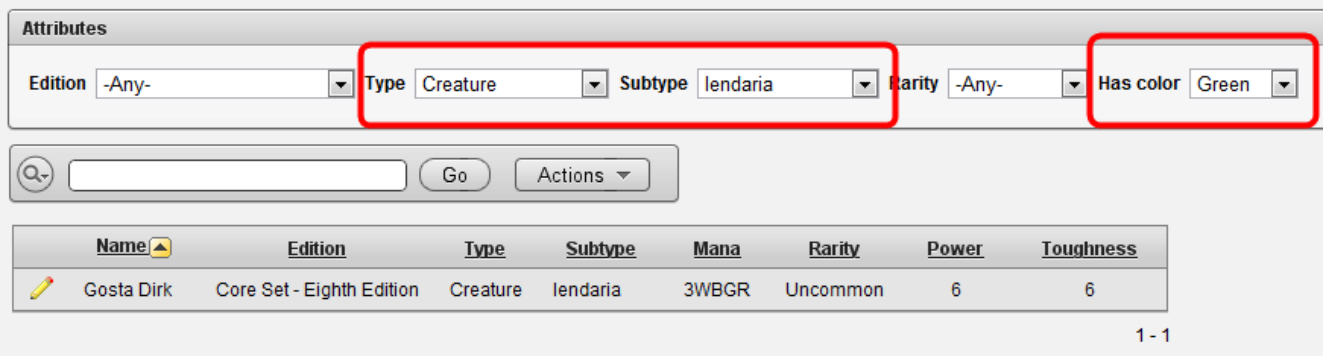
Ao carregar, por exemplo, no tipo de carta *Creature,* na página *Normal Cards* irão aparecer só com cartas do tipo *Creature*.



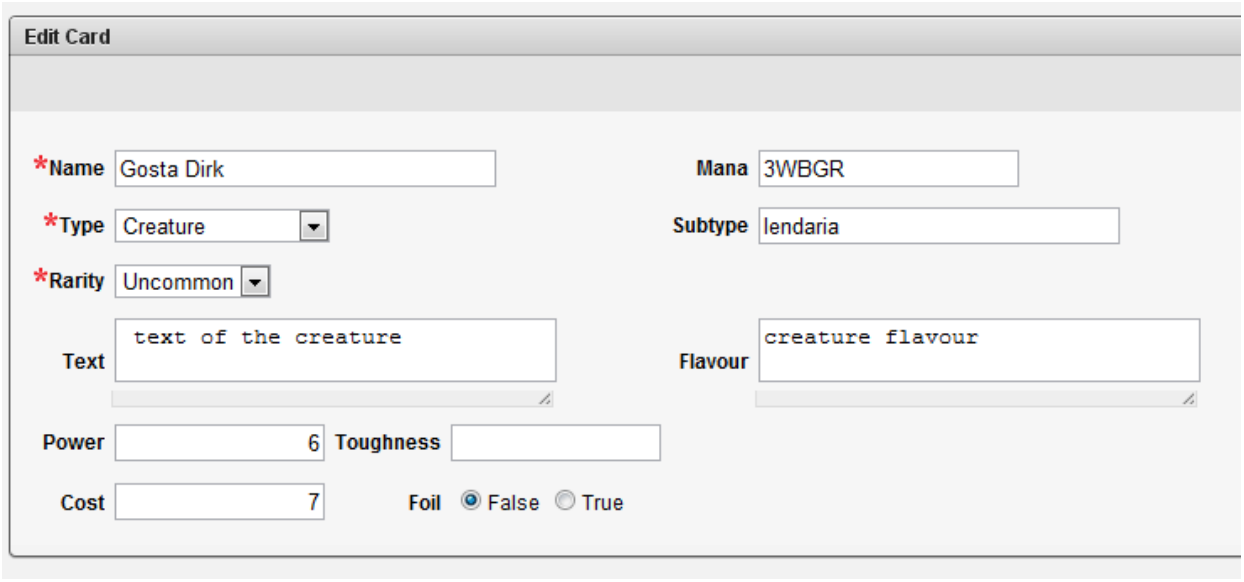
Se carregarmos na edição *Mirage*, a página *Normal Cards* aparecerá da seguinte forma:



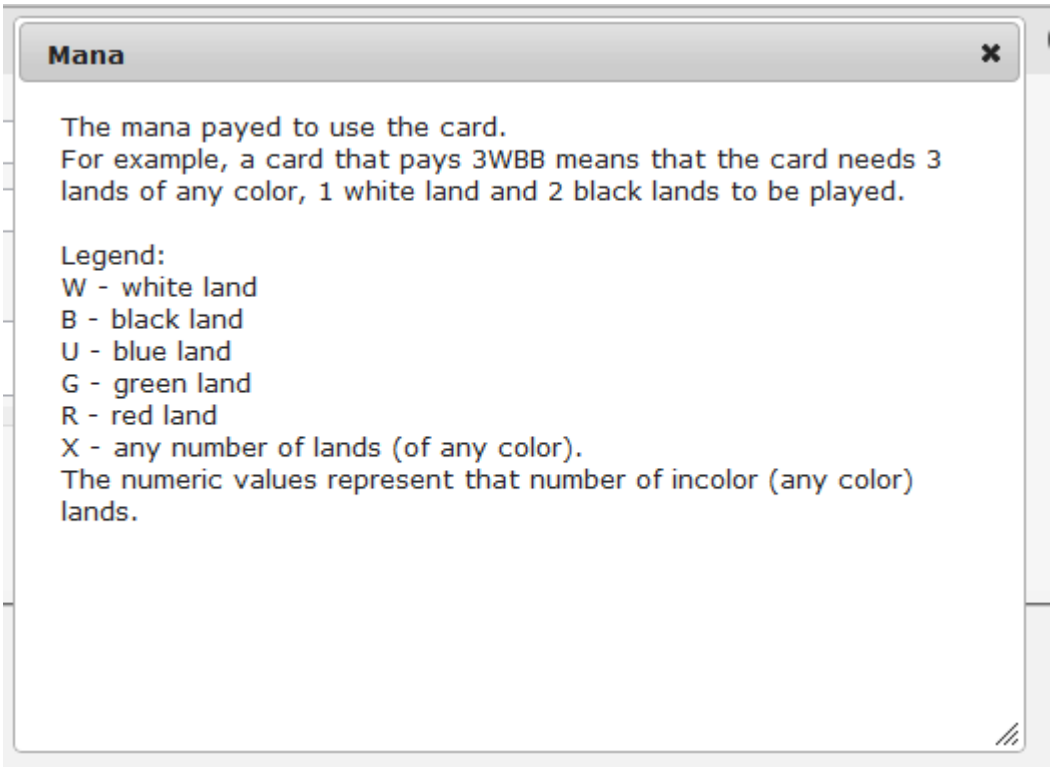
Dentro da página *Normal Cards* podemos seleccionar diferentes valores nas diversas *select list* em simultâneo, para filtrar as cartas. Se o utilizador procurar cartas do tipo criatura, do subtipo lendária e que sejam verdes, a listagem das cartas ficará da seguinte forma:



Ao carregar no lápis para editar a carta o seguinte formulário será apresentado 

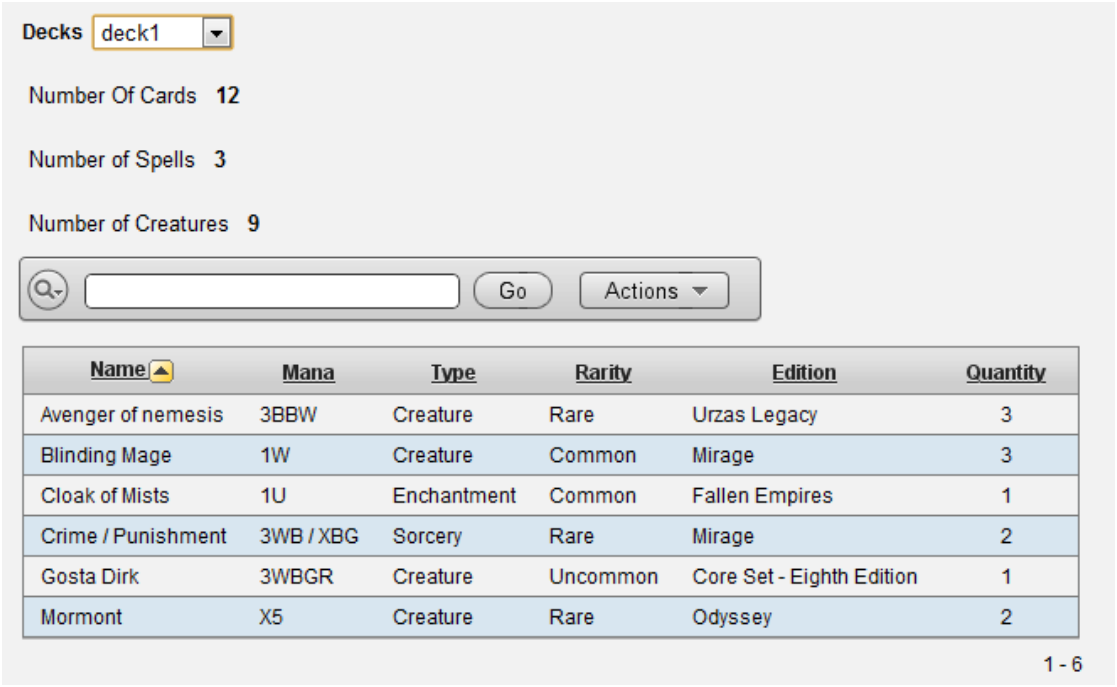


Aqui pode seleccionar-se o tipo de carta e a raridade a partir de *select list* e se a carta é ou não brilhante a partir de um *radio group*. Se o utilizador estiver com dúvidas em relação ao que escrever nos campos basta carregar em cima do nome do atributo e o *Help Text* aparecerá:

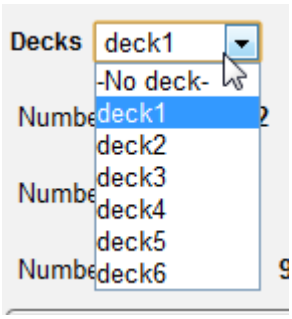


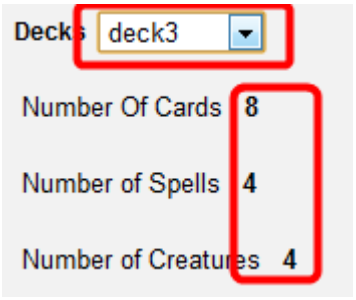
Se o utilizador colocar dados incorrectos, ao submeter os dados será alertado em relação a isso:

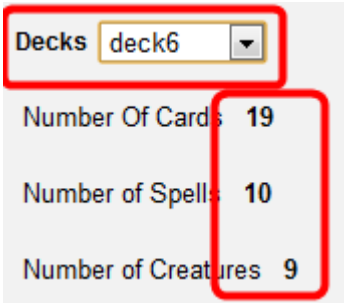
Na página principal se carregarmos em *Report Decks* a seguinte página será apresenta com o *select list* seleccionado no deck1:



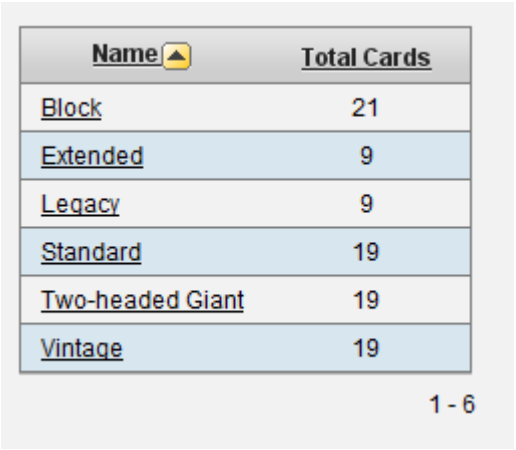
Nesta página é apresenta, em cima, uma *select list* para escolhermos o *deck* pretendido e por baixo da *select list* aparecem informações sobre o número de cartas do deck:



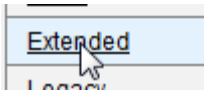


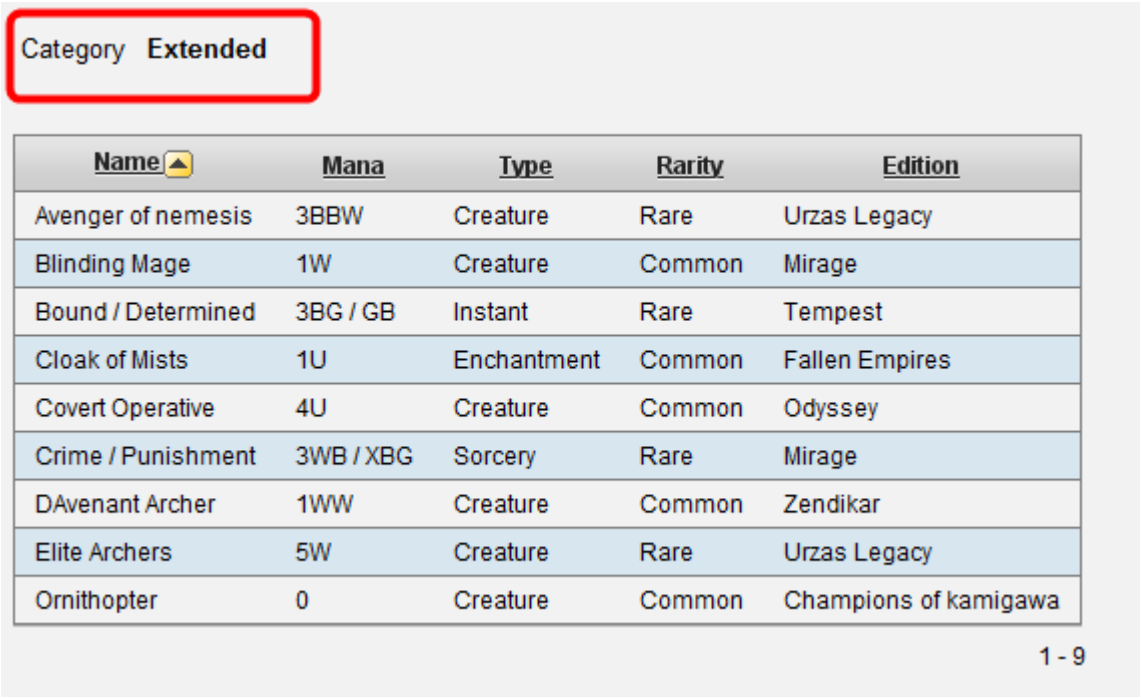


Na página principal, se carregarmos em *Report Categories,* a seguinte página será apresentada:



Na coluna *Total Cards* é mostrado o total de cartas que podem jogar na respectiva categoria, e se carregarmos na categoria uma nova página será aberta, na qual podemos ver as cartas que podem jogar nessa categoria:





Category **Extended**

Name	Mana	Type	Rarity	Edition
Avenger of nemesis	3BBW	Creature	Rare	Urzas Legacy
Blinding Mage	1W	Creature	Common	Mirage
Bound / Determined	3BG / GB	Instant	Rare	Tempest
Cloak of Mists	1U	Enchantment	Common	Fallen Empires
Covert Operative	4U	Creature	Common	Odyssey
Crime / Punishment	3WB / XBG	Sorcery	Rare	Mirage
DAvenant Archer	1WW	Creature	Common	Zendikar
Elite Archers	5W	Creature	Rare	Urzas Legacy
Ornithopter	0	Creature	Common	Champions of kamigawa

1 - 9

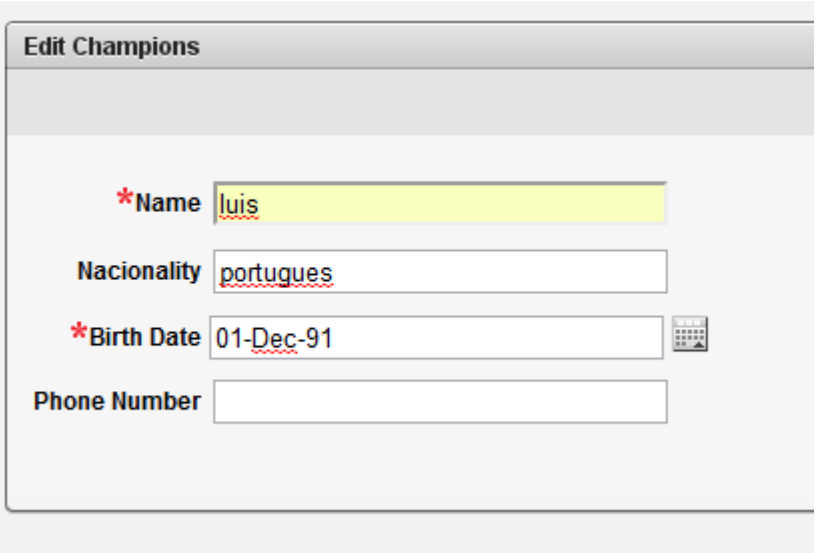
Na parte superior é mostrada uma nota a relembrar que categoria foi escolhida.

Na página principal, se carregarmos em *Report of Champions,* a seguinte página será apresentada:

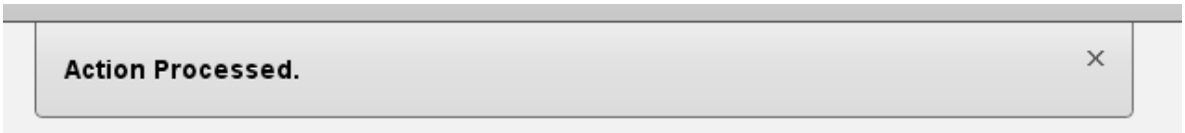
Edit	Name	Nacionality	Birth Date	Phone Number
	Vicent patrick	frances	21-DEC-73	-
	bill gates	estados unidos da america	28-OCT-55	-
	catarina albino	portugues	06-NOV-91	911000000
	luis silva	portugues	04-SEP-91	-
	ruben cachucho	portugues	12-SEP-90	910000000
	santiago miranda	espanhol	21-DEC-83	933114000230

1 - 6

Nesta página vemos os campeões existentes na base de dados. Se carregarmos no botão *Create* podemos criar um novo campeão.



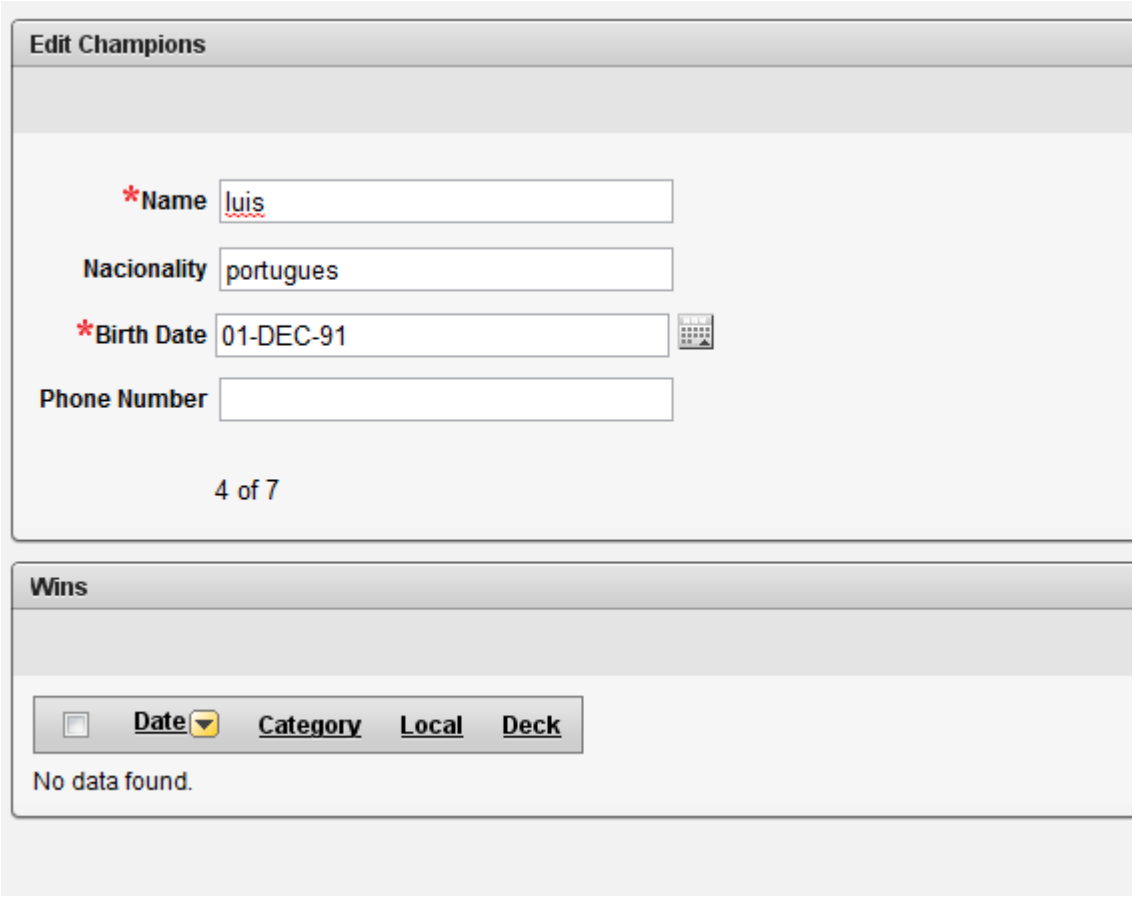
Nota: temos de colocar uma data de nascimento onde o campeão tenha mais de 12 anos.



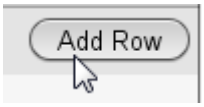
Edit	Name	Nacionality	Birth Date	Phone Number
	Vicent patrick	frances	21-DEC-73	-
	bill gates	estados unidos da america	28-OCT-55	-
	catarina albino	portugues	06-NOV-91	911000000
	luis	portugues	01-DEC-91	-
	luis silva	portugues	04-SEP-91	-
	ruben cachucho	portugues	12-SEP-90	910000000
	santiago miranda	espanhol	21-DEC-83	933114000230

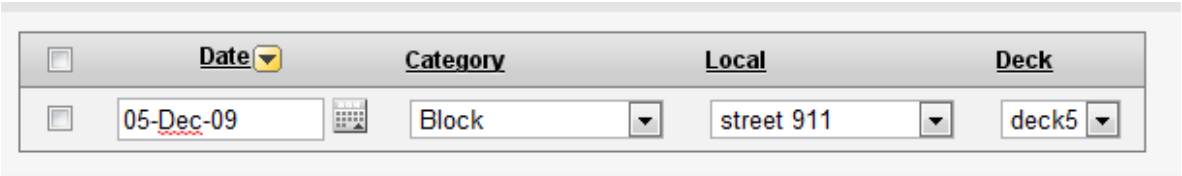
1 - 7

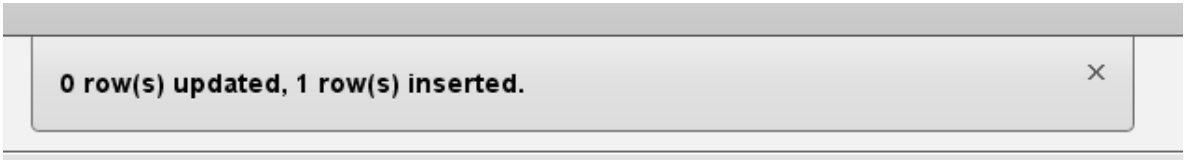
Ao carregarmos para editar os torneios ganhos por este campeão o seguinte formulário aparecerá:



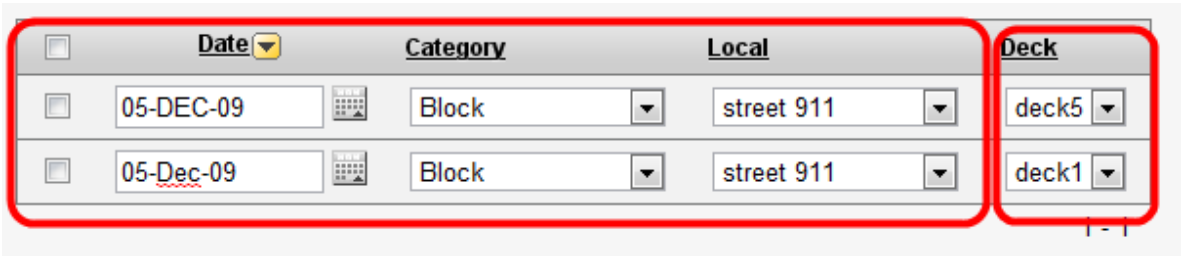
Se carregarmos em botão *Add Row,* podemos adicionar um torneio ganho por este campeão:

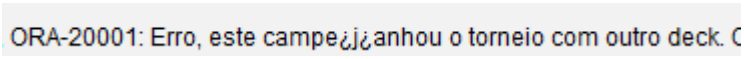




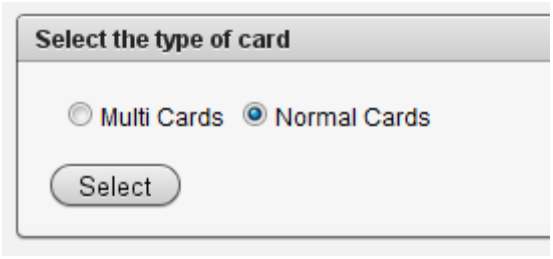


Ao adicionarmos dados inconsistentes seremos avisados:

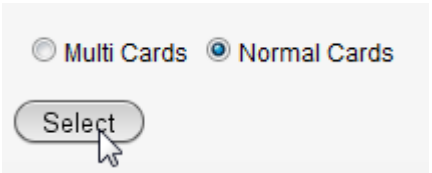




Na página principal se carregarmos em *Create / Edit Cards* a seguinte página será apresentada:



Nesta página aparece um *radio group* com a opção de escolher ou *normal cards* ou *Multi Cards* para editarmos e/ou criamos, por definição aparece as *Normal Cards* seleccionada. Se carregarmos no botão *Select* com a opção *Normal Cards* a seguinte página é apresentada:

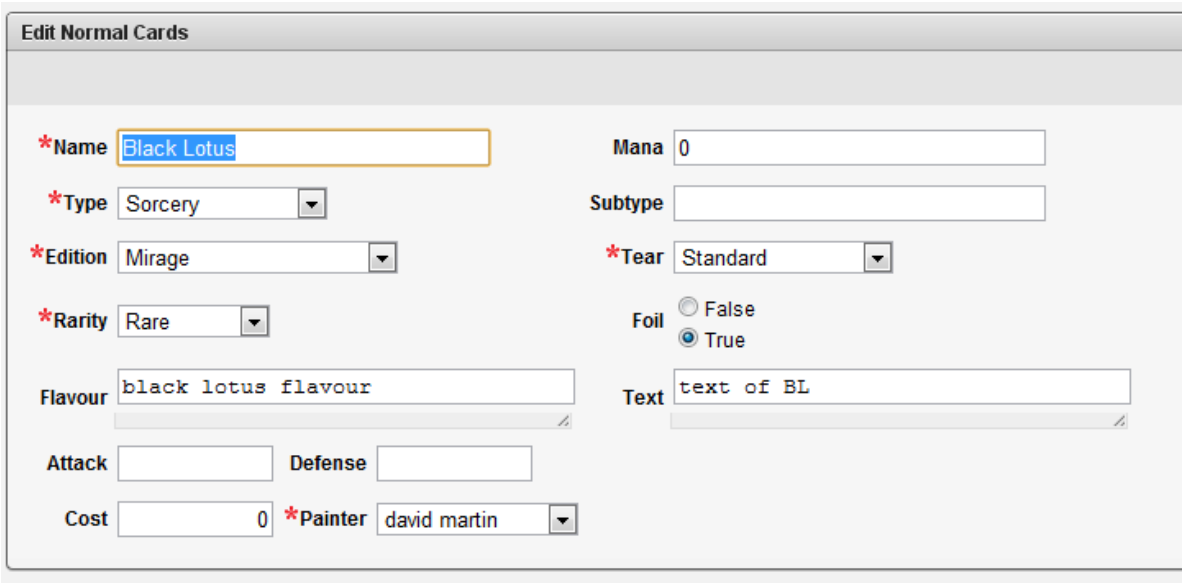


Name	Mana	Type	Subtype	Rarity	Edition	Tear	Cost	Power	Toughness	Foil	Painter
Arrest	2W	Enchantment	-	Uncommon	Zendikar	Block Constructed	3	-	-	0	dan frazier
Avenger of nemesis	3BBW	Creature	avenger	Rare	Urzas Legacy	Extended	6	7	5	1	ron spears
Black Lotus	0	Sorcery	-	Rare	Mirage	Standard	0	-	-	1	david martin
Blinding Mage	1W	Creature	Human-Wizzard	Common	Mirage	Extended	2	1	2	0	greg hildebrandt
Bog Walker	2BB	Creature	zombie	Uncommon	Fallen Empires	Standard	4	3	4	0	greg hildebrandt
Cloak of Mists	1U	Enchantment	-	Common	Fallen Empires	Extended	2	-	-	0	greg hildebrandt
Covert Operative	4U	Creature	Wizzard	Common	Odyssey	Extended	5	3	2	0	greg hildebrandt
Cowardice	3BB	Creature	dark elf	Rare	Core Set - Eighth Edition	Standard	5	4	5	0	glenn fabry
Crown of Awe	1W	Enchantment	-	Common	Core Set - Eighth Edition	Standard	2	-	-	0	glenn fabry
DAvenant Archer	1WW	Creature	Archer	Common	Zendikar	Extended	3	1	2	0	dan frazier
Elite Archers	5W	Creature	Soldier	Rare	Urzas Legacy	Extended	6	3	3	1	ron spears
Gosta Dirk	3WBGR	Creature	lendaria	Uncommon	Core Set - Eighth Edition	Standard	7	6	6	0	jonhn snow
Mahamoti Djinn	4RB	Creature	Djinn	Uncommon	Tempest	Block Constructed	6	5	6	1	david martin
Mormont	X5	Creature	lendaria	Rare	Odyssey	Standard	5	8	6	1	jonhn snow
Ornithopter	0	Creature	Thpter	Common	Champions of kamigawa	Extended	0	0	2	0	ron spears

1 - 15

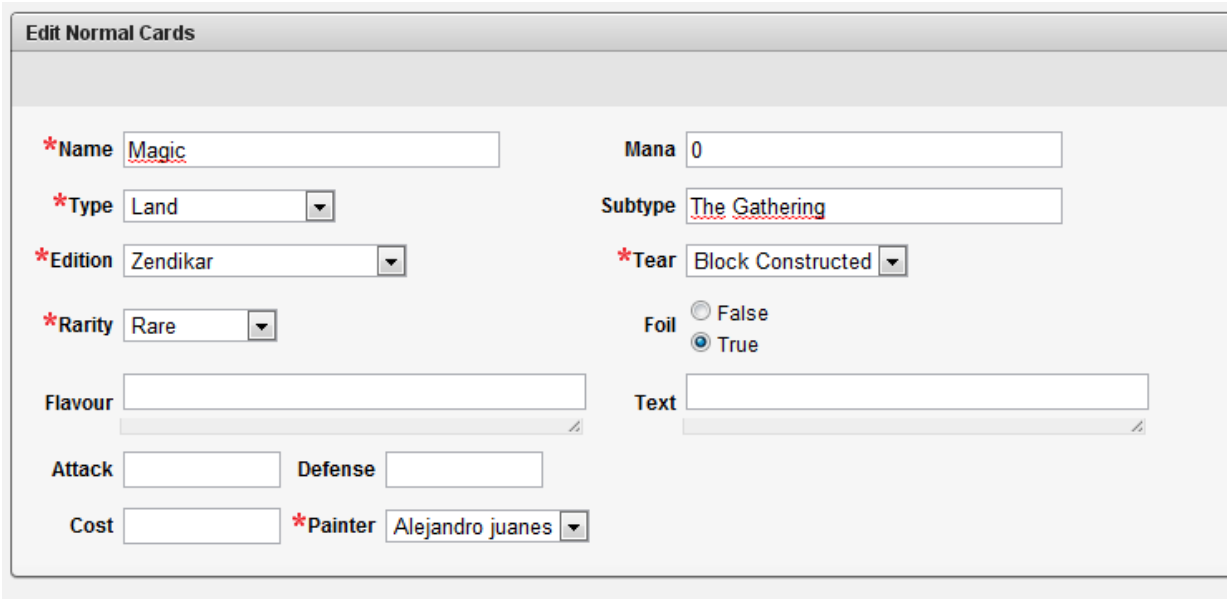
Nesta página é mostrada toda a informação referente às *normalcards*, onde as chaves externas aparecem com os seus valores substituídos por valores mais legíveis.

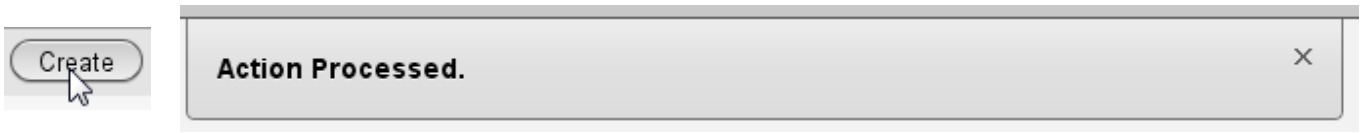
Ao carregarmos para editar uma carta, aparece um formulário similar ao formulário que aparece na página *Normal Cards*, mas este contém mais atributos que podemos editar, como o seu pintor e o seu *tear*:

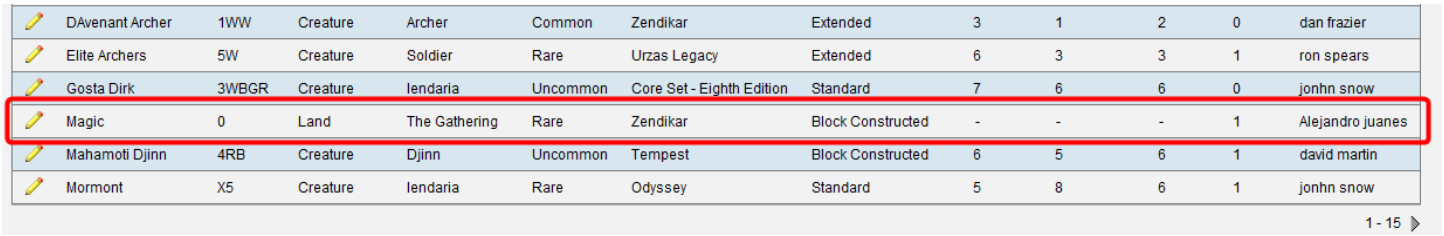


Esta página também contém *Help Text* , que descrevem os valores esperados para os vários atributos e que podem ser usados pelos utilizadores em caso de dúvida.

Se o utilizador se enganou na carta a editar, mesmo que já tenha alterado alguns dados basta carregar no botão *Cancel* para que as alterações não serão gravadas, e a página anterior será apresentada. Nesta podemos criar uma nova carta carregando no botão *Create*:



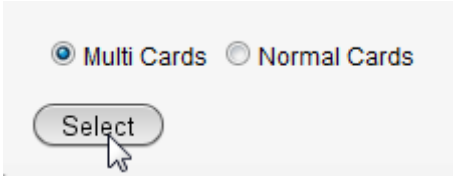




DAvenant Archer	1WW	Creature	Archer	Common	Zendikar	Extended	3	1	2	0	dan frazier
Elite Archers	5W	Creature	Soldier	Rare	Urzas Legacy	Extended	6	3	3	1	ron spears
Gosta Dirk	3WBGR	Creature	Iendaria	Uncommon	Core Set - Eighth Edition	Standard	7	6	6	0	jonhn snow
Magic	0	Land	The Gathering	Rare	Zendikar	Block Constructed	-	-	-	1	Alejandro juanes
Mahamoti Djinn	4RB	Creature	Djinn	Uncommon	Tempest	Block Constructed	6	5	6	1	david martin
Mormont	X5	Creature	Iendaria	Rare	Odyssey	Standard	5	8	6	1	jonhn snow

1 - 15

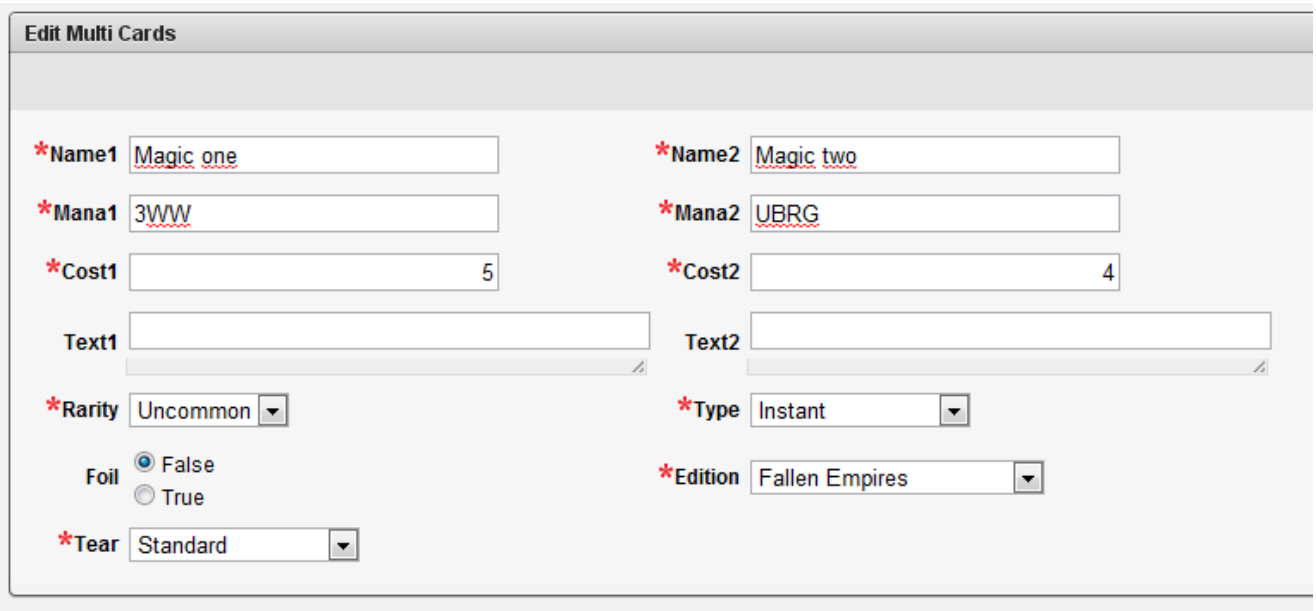
Se tivéssemos escolhido editar as *multi cards* a seguinte página era apresentada:

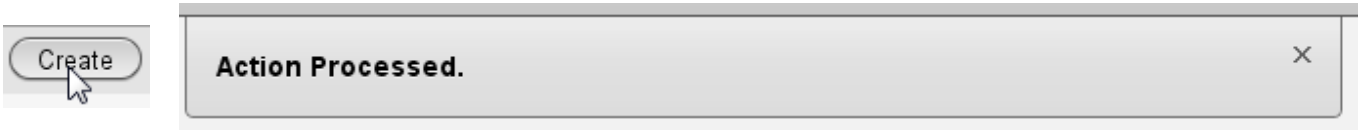


	Name1	Name2	Mana1	Mana2	Type	Rarity	Edition	Cost1	Cost2	Foil	Tear
	Angels Killer	Bog phantom	1BB	3B	Creature	Uncommon	Core Set - Eighth Edition	3	4	0	Standard
	Bound	Determined	3BG	GB	Instant	Rare	Tempest	5	2	0	Extended
	Crime	Punishment	3WB	XBG	Sorcery	Rare	Mirage	5	3	0	Extended

1 - 3

Nesta página temos as mesmas opções que na página de edição das *normalcards*, mas com os dados referentes às *multicards*. Se quisermos criar uma nova *multicard* carregamos no botão *Create* e inserirmos os dados:





	Name1	Name2	Mana1	Mana2	Type	Rarity	Edition	Cost1	Cost2	Foil	Tear
	Angels Killer	Bog phantom	1BB	3B	Creature	Uncommon	Core Set - Eighth Edition	3	4	0	Standard
	Bound	Determined	3BG	GB	Instant	Rare	Tempest	5	2	0	Extended
	Crime	Punishment	3WB	XBG	Sorcery	Rare	Mirage	5	3	0	Extended
	Magic one	Magic two	3WW	UBRG	Instant	Uncommon	Fallen Empires	5	4	0	Standard

1 - 4

4. Considerações finais

A nível da Base de Dados pensamos ter implementado tudo o que tínhamos descrito nos objectivos. Pensamos que a nossa Base de Dados modela todos os aspectos da realidade necessários relativos a torneios de cartas magic.

Inicialmente, tínhamos em mente não deixar apagar quase nada da BD pois queríamos manter o histórico completo, mas após a discussão desta ideia com o professor, decidimos que tal não fazia sentido pois os enganos acontecem. Decidimos então que só iríamos deixar os utilizadores apagarem cartas se não pertencerem a nenhum deck.

Tendo em conta as simplificações que fizemos para não tornar a BD demasiado complexa para este projecto, consideramos que a BD cumpre todos os objectivos e metas propostas para este trabalho e para uma BD com a mesma finalidade.

Concluindo, com a resolução deste trabalho percebemos a importância de desenhar primeiramente um modelo de entidade e relação consistente e não redundante para uma melhor implementação da BD, mais coerente, mais simples e mais optimizada.

5. Glossário

Para clarificar o significado dos termos específicos usados no âmbito do tema da nossa base de dados, colocamos aqui a respectiva definição e alguns exemplos:

- Tipo - tipo de carta (terreno, criatura, mágica instantânea, etc.);
- Subtipo – *wizard*, elfo, arcana, etc.;
- Raridade - rara, incomum ou comum;
- Cor – branco, azul, preta, vermelha, verde e artefacto (incolor);
- *Foil* – indicação sobre o facto de a carta ser ou não brilhante;
- *Mana* - representa os tipos de terrenos que um jogador tem de "pagar" para colocar a carta em campo;
- Ataque – valor do dano causado aquando de um ataque ou bloqueio (só as cartas do tipo criatura têm esta propriedade);
- Defesa - representa quanto dano é necessário para destruir a carta (só as cartas do tipo criatura têm esta propriedade);
- *Flavour* - frase caracterizadora do estilo/imagem da carta.
- Custo – total de mana necessária para colocar a carta em jogo;

6. Anexos

6.1 Anexo A

Figura 1: localização dos atributos de uma carta Magic.

6.2 Anexo B

Figura 2: Exemplo de uma *multicard*.