

# Jogos de tabuleiro

## Relatório 3ª Fase



Universidade do Porto

Faculdade de Engenharia

**FEUP**

Mestrado Integrado em Engenharia Informática e  
Computação

Base de dados

### **Grupo 601:**

Hugo Ari Rodrigues Drumond — 201102900

Ricardo Jorge Matos Figueiredo — 201100687

Gustavo Assis Freitas — 200602187

Faculdade de Engenharia da Universidade do Porto  
Rua Roberto Frias, 4200-65 Porto, Portugal

1 de Junho de 2014

## 1 Contexto

Esta base de dados destina-se a um Salão de jogos que organiza jogos de Tabuleiro e foi desenvolvida, inicialmente, pensando num só jogo, o xadrez. Posteriormente, então, foi feita uma generalização. Em suma, são guardados os dados dos jogadores e dos torneios de uma dada temporada. Isto, possibilitará aos utilizadores da base de dados: rigor ao planejar eventos, versatilidade, facilidade de registo, entre outros. Por exemplo, se eu quisesse organizar um torneio, em condições, iria ter saber à priori: os escalões, o jogo a que diz respeito, a temporada, as equipas inscritas e os patrocinadores. E registá-los em algum sítio para que depois possa associá-los, direta ou indiretamente, a uma partida e uma partida a equipas. A nossa base de dados tem como único propósito tornar esse {pré,pós}registo trivial.

## 2 Conceitos Principais

Num torneio de jogos de tabuleiro podem haver várias partidas entre duas ou mais equipas num dado escalão de um torneio. As partidas podem ocorrer em diversos sítios, têm de ser reguladas por um ou mais árbitros qualificados para o jogo em disputa e os resultados devem ser guardados. Cada equipa é formada por um ou mais jogadores. Podem haver vários torneios exatamente iguais, independentemente de quaisquer condicionantes.

## 3 Passagem ao modelo relacional e normalização

A transição do diagrama de classes para o schema da base de dados foi feita sem grandes problemas. Optámos por separar a generalização em duas tabelas segundo o estilo orientado a objetos. Porque ao fazer isto, ficamos no ponto de equilíbrio entre poupar nas ligações de tabelas e nos recursos de armazenamento. Uma relação, menos ligações embora haja desperdício de armazenamento; Três relações, mais ligações e mesmo uso de recursos de armazenamento. A restrição de especialização do tipo complete e disjoint, visto que a classe base, Pessoa, só serve de template (classe virtual) e devido à impossibilidade de uma Pessoa ser árbitro e jogador.

A única ternária no nosso uml foi decomposta em EquipaPatrocinadorTorneio e a classe de associação em EquipaPartida. Mais as tabelas de "apoio". Na tabela Jogador o idPaís->País tem o significado de Nacionalidade e idCidade->Cidade é um "apontador" para uma cidade que por sua vez está associada a um País, e representa a cidade onde ele reside. O telefone e a extensão podem não estar ligados nem ao País nem à morada de uma Pessoa, por isso não servem de determinante para nenhum desses atributos, porque uma Pessoa pode viver num País, ser de outro e ter um contacto registado ainda noutro País.

As relações muitos para muitos foram decompostas da maneira tradicional, uma tabela para cada relação mais uma de interligação. No caso das um para muitos, criámos uma tabela para cada classe e um campo na de multiplicidade muitos que "aponta" para a tabela de multiplicidade um.

## Schemas

**Formato**(idFormato, nome)  
**Escalao**(idEscalao, nome)  
**TipoJogo**(idTipoJogo, nome)  
**Patrocinador**(idPatrocinador, nome)  
**Extensao**(idExtensao, codigo)  
**Pais**(idPais, nome)  
**Equipa**(idEquipa, nome, abreviatura)  
**Cidade**(idCidade, nome, idPais->Pais)  
**Jogador**(idJogador, nome, codigoPostal, dataNascimento, numeroAndar, rua, telefone, idPais->Pais, idCidade->Cidade, idExtensao->Extensao, email)  
**Árbitro**(idÁrbitro, nome, codigoPostal, dataNascimento, numeroAndar, rua, telefone, idPais->Pais, idCidade->Cidade, idExtensao->Extensao, observacoes)  
**LocalEncontro**(idLocalEncontro, idCidade->Cidade, idExtensao->Extensao, codigoPostal, rua, telefone)  
**Torneio**(idTorneio, idTipoJogo->TipoJogo, nome, temporada, idFormato->Formato)  
**Partida**(idPartida, idTorneio->Torneio, idLocalEncontro->LocalEncontro, dataInicio, duracao, idEscalao->Escalao)  
**JogadorEquipa**(idEquipa->Equipa, idJogador->Jogador)  
**ArbitroTipoJogo**(idArbitro->Arbitro, idTipoJogo->TipoJogo)  
**ArbitroPartida**(idArbitro->Arbitro, idPartida->Partida)  
**EquipaPartida**(idEquipa->Equipa, idPartida->Partida, posicao, resultado)  
**EquipaPatrocinadorTorneio**(idEquipa->Equipa, idPatrocinador->Patrocinador, idTorneio->Torneio)

A nossa base de dados encontra-se na 3ª Forma Normal, visto que respeita a 1ª, 2ª, 3ª formas normais. Deste modo a nossa base de dados evita algumas anomalias: redundância de dados, atualização de campos, campos sinónimos, etc. Não continuámos a normalizar para além da 3ª Forma Normal porque isto tornaria a nossa base de dados mais lenta devido à quantidade de ligações, mais difícil de pesquisar e por não ser exigido pelos docentes. Todas as nossas relações estão na 1ª Forma Normal porque são respeitadas as regras das tabelas e porque em cada relação está definida uma chave primária. Também está na 2ª visto que nenhum subconjunto das chaves primárias determina um atributo não chave, as nossas relações só têm uma chave ou são todas chaves portanto respeitam de imediato esta Forma Normal. Excepto a classe de associação que é necessário identificar se um subconjunto da chave identifica um atributo não chave, que não se verifica, portanto está na segunda Forma Normal. Para respeitar a 3ª Forma Normal não podem haver dependências funcionais transitivas,  $A \rightarrow B \ \&\& \ B \rightarrow C$ , que não acontece na nossa bases de dados. A única razão para a nossa base de dados não respeitar a BCNF restringe-se às relações Árbitro e Jogador, porque existe um determinante em cada uma das relações que não é chave candidata, a dependência funcional  $idCidade \rightarrow Cidade, rua \rightarrow codigoPostal$ . Isto é, se partirmos do ponto de vista que é possível determinar o código postal sabendo a Rua, a Cidade e o País em causa (o país está relacionado com Cidade). Cada Cidade tem um nome, um "apontador" para um País e um código único que serve de chave primária.

## 4 Linguagem de Definição de dados e Restrições

O ficheiro de criação da base de dados chama-se boardDb.sql, e para além do óbvio, pode-se encontrar as seguintes restrições:

**Restrições de valor:** NOT NULL e de limites de atributos.

**Restrições entre atributos:** não foi utilizado porque não temos membros em que faça sentido utilizar este tipo de restrição.

**Restrições de chave externa:** Em vez de ser rejeitada a modificação de uma tabela que é referenciada noutra decidimos que: caso houvesse um delete do destino, o "apontador" devia ser NULO e numa atualização todos os valores antigos deviam ser alterados em cascata, ON DELETE SET NULL && ON UPDATE CASCADE .

**Triggers:** O ficheiro com os triggers chama-se triggers.sql. E lá podemos encontrar dois pares de gatilhos que impossibilitam a inserção ou atualização de informação inválida.

- Um arbitro só pode arbitrar uma partida se tiver competências para tal (Update e Insert)
- Um jogador só pode estar numa equipa num torneio (Update e Insert)

E obviamente que foram definidas chaves que são um tipo de restrição.

## 5 Linguagem de Manipulação de dados - Inserção

Todas as tabelas foram preenchidas com informação decente e de tamanho apreciável de modo a tornar as pesquisas mais interessantes. Temos uma média de inserts perto das 9 linhas/tabela. O ficheiro com os inserts chama-se inserts.sql.

## 6 Linguagem de Manipulação de dados - Consulta

O ficheiro com as consultas chama-se queries.sql, e lá encontram-se as seguintes perguntas à base de dados:

Principais

- Encontrar o nome, o email e a nacionalidade de todos os jogadores que participaram num dado torneio num dado escalão
- Encontrar o jogador com mais pontos
- Encontrar a pontuação de todos os jogadores de uma equipa para todas as partidas que eles realizaram
- Encontrar o jogador com mais pontos de uma Equipa
- Encontrar a variância da pontuação dos jogadores de uma equipa
- Encontrar a equipa com mais pontos para um tipo de jogo e escalão

- Encontrar a equipa com mais pontos
- Encontrar a equipa com mais pontos para um dado torneio num dado escalão

Outros

- Ver a duração média dos jogos para um dado tipo jogo
- Ver qual é o país mais comum
- Encontrar o nome, o email e a nacionalidade de todos os jogadores que jogam num tipo de jogo num dado escalão.
- Encontrar o nome, o email e a nacionalidade de todos os jogadores que participaram num dado torneio com um dado nome( podem haver vários).
- Encontrar todos os jogadores pertencentes às equipas, e mostrar o idE-quipa; nome, email e nacionalidade do jogador.
- Encontrar todos os jogadores de um dada equipa
- Contar o número de equipas em que os jogadores estão inscritos
- Pontuação total por jogador

## 7 Diagrama de classes UML melhorado

