**<u>Verificar o segmento “Notas Importantes” na página 6.
Essencial para a avaliação do trabalho.</u>**

PROJETO DE AVALIAÇÃO
BASES DE DADOS

1º SEMESTRE, 2022/2023

Carolina Viana, up202108802

Guilherme Monteiro, up202108668

Sofia Sá, up202108676

DIAGRAMA UML (1ª entrega)

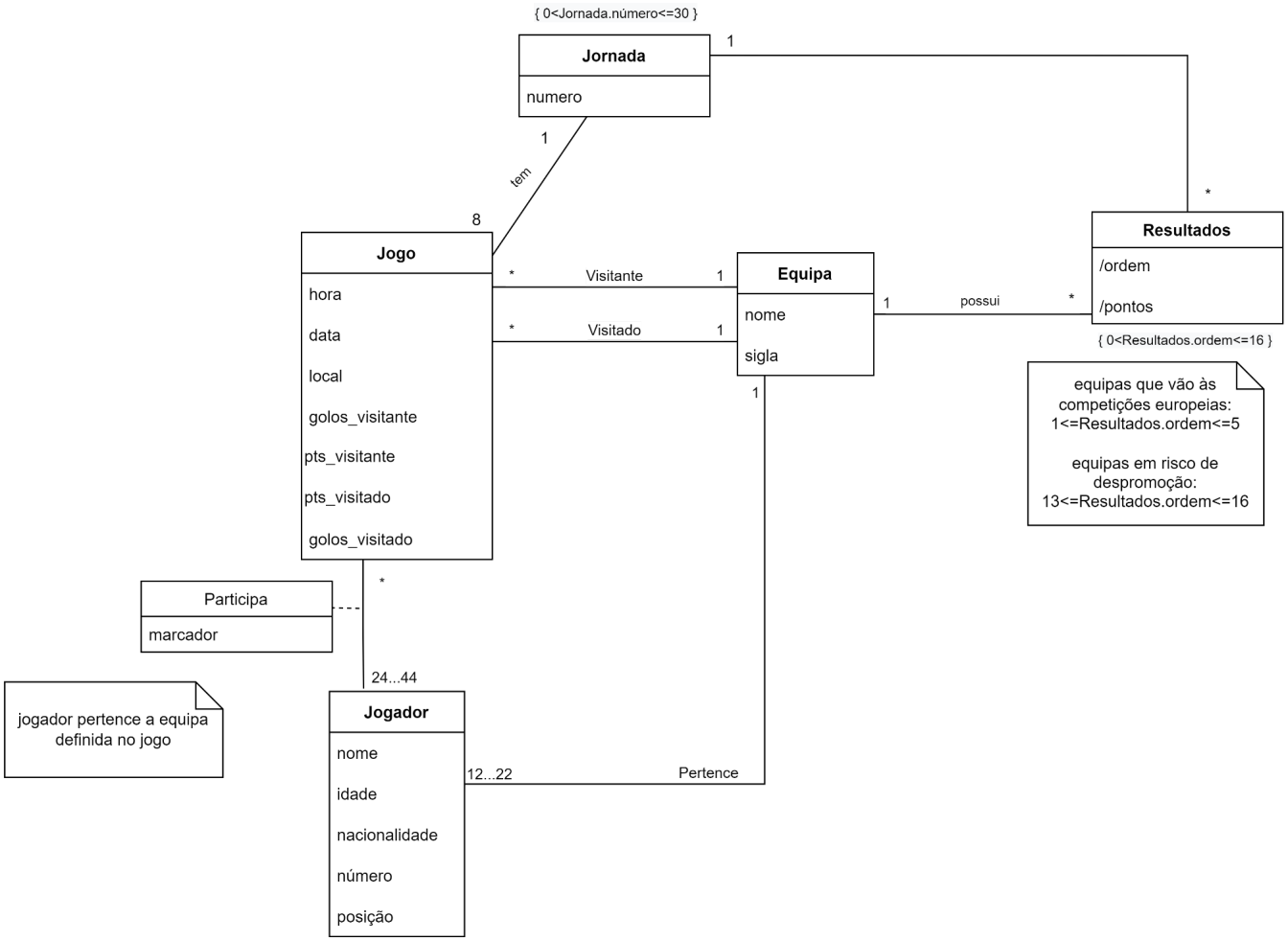
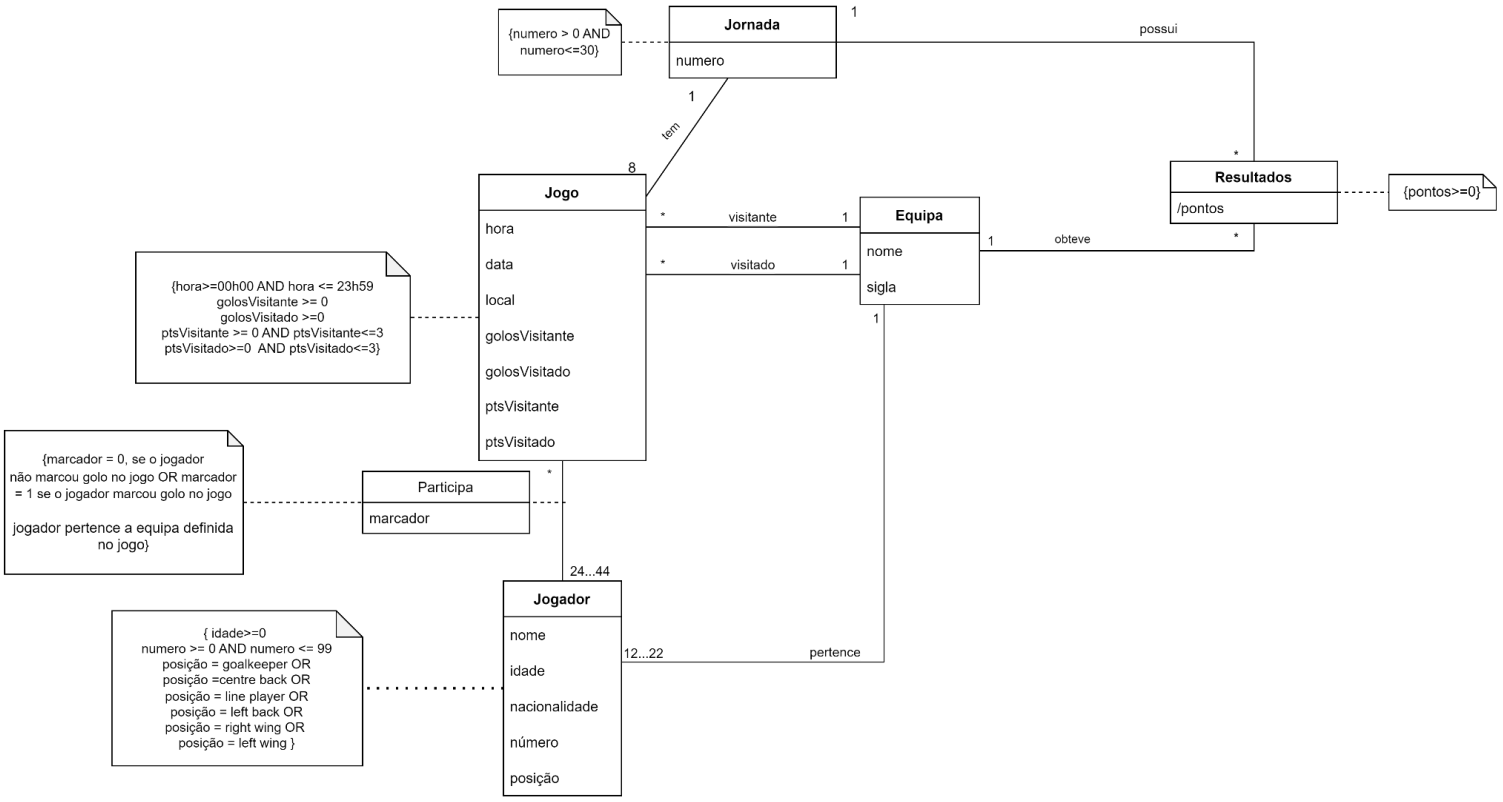


DIAGRAMA UML (2ª entrega)



Alterações relativamente ao UML da primeira entrega:

- Correção no alinhamento dos atributos das classes.
- Conversão dos nomes dos atributos para camelCase.
- Atribuição de nome às associações restantes.
- Adição de novas restrições às várias classes

Nota Relevante:

- A restrição associada à classe de associação “Participa” - “jogador pertence a equipa definida no jogo” - não é obedecida no povoamento inicial da Base de Dados. Assim, resolvemos a questão implementando um Trigger, que garante que a restrição é obedecida.

CONTEXTO

Para este trabalho, escolhemos e procuramos informações sobre o Campeonato Português de Andebol de Séniores Masculinos da I Liga da época 2021/2022.

Em primeiro lugar, para uma contextualização acerca do nosso diagrama UML, iremos explicar as multiplicidades que incluímos.

Este campeonato é composto por 30 jornadas e 16 equipas. Cada jornada é composta por 8 jogos (todas as equipas têm um jogo por jornada).

Não existe número definido de jogadores por equipa (nas equipas portuguesas, o número de jogadores varia de equipa para equipa), no entanto, pode haver um máximo de 22 jogadores inscritos numa equipa sénior e, como são necessários 12 jogadores para ir a um jogo, consideramos esse como número mínimo. Seguindo essa linha de pensamento, percebemos que num jogo, podem intervir 24 a 44 jogadores (de ambas as equipas).

Cada jogador pertence apenas a uma equipa e pode participar em vários jogos ao longo da época.

Em cada jogo intervém uma equipa visitante e uma equipa visitada. No entanto, cada equipa pode ir a múltiplos jogos.

Cada equipa vai obtendo vários resultados ao longo da época, no entanto, cada resultado se refere apenas a uma equipa. O mesmo acontece com a jornada (cada jornada engloba vários resultados; todavia, cada resultado é apenas referente a uma jornada).

Terminada a explicação acerca das multiplicidades escolhidas, iremos proceder a uma exposição geral sobre o nosso diagrama UML, explicitando algumas das nossas escolhas.

Cada jornada tem um número da jornada que, como explicado anteriormente, pode ir de 1 a 30.

A cada jogo tiramos informações sobre a hora, local e data. Optamos por dividir o resultado, mostrando os golos das duas equipas (visitante e visitada) em separado. Fizemos isso de maneira a simplificar a forma de obter o número de pontos ganhos por cada equipa nesse determinado jogo (ptsVisitante e ptsVisitado). Por cada vitória, a equipa recebe 3 pontos, por cada empate 2 pontos e por cada derrota 1 ponto.

Um jogador tem um nome, idade, nacionalidade, número e posição e, ao participar num jogo (classe Participa), pode ser marcador ou não. Expomos isso na Base de Dados através do atributo “marcador” que funciona como um “bit sinal”: se for 1, o jogador marcou pelo menos um golo durante o jogo. Se for 0, não marcou nenhum. Tomamos

essa decisão devido ao requisito no enunciado “Esta base de dados deve responder ao lançamento de [...] marcadores dos golos”.

Cada equipa tem um nome e uma sigla.

A cada equipa corresponde também um Resultado por jornada. Cada resultado expõe exatamente quantos pontos tem a equipa (atributo pontos), bem como a sua posição na tabela geral (atributo ordem), isto é se vai em 1º lugar, 2º, 3º, por aí em diante.

Estes campos serão calculados dinamicamente pela base de dados, através dos dados que inserimos para as outras tabelas. De momento, ainda não temos esse conhecimento, pelo que optamos por, nos passos seguintes de criação e povoamento das tabelas em SQLite, proceder apenas à criação da tabela Resultados de uma maneira muito simples, não a povoando ainda, conforme o referido no enunciado “As restrições que necessitarem de um gatilho para serem implementadas, devem ser deixadas para a Entrega II do projeto.”

No diagrama UML referimos apenas as informações mais básicas, como, por exemplo, o facto da ordem poder ir apenas da 1ª posição até à 16ª. Contudo, futuramente, iremos calcular os pontos cumulativos de cada equipa em cada jornada, somando os pontos acumulados até então com os pontos obtidos no jogo dessa jornada (iremos explorar mais sobre isto, numa fase mais à frente do relatório).

Por fim, para saber “as equipas em condições de ir às competições europeias e em risco de despromoção”, basta observar o atributo “ordem” da classe Resultados. As primeiras 5 equipas (ordem 1, 2, 3, 4 e 5) estão em condição de ir às diversas competições europeias. As últimas 4 equipas (ordem 13, 14, 15 e 16) estão em risco de despromoção.

FONTE: Federação de Andebol de Portugal

https://portal.fpa.pt/wp-content/uploads/2022/07/PO01-2022-2023.pdf

MODELO RELACIONAL

Jornada (idjornada, número)

Jogo (idjogo, hora, data, local, golosVisitante, golosVisitado, ptsVisitante, ptsVisitado, idjornada→Jornada, idvisitante→Equipa, idvisitado→Equipa)

Equipa (idequipa , nome, sigla)

Resultados (idresultados, pontos, idjornada→Jornada, idequipa→Equipa)

Jogador (idjogador, nome, idade, nacionalidade, número, posição, idequipa→Equipa)

Participa (idjogador→Jogador, idjogo → Jogo, marcador)

ANÁLISE DE DEPENDÊNCIAS FUNCIONAIS E FORMAS NORMAIS

FDs:

- Jornada (idjornada, número)

  FD: { {idjornada} → {número}

  {número} → {idjornada} }

  Formas: BCNF: sim

  3NF: sim

- Jogo (idjogo, hora, data, local, golosVisitante, golosVisitado, ptsVisitante, ptsVisitado, idjornada→Jornada, idvisitante→Equipa, idvisitado→Equipa)

  FDs: { {idjogo} → {hora, data, local, golosVisitante, golosVisitado, ptsVisitante, ptsVisitado, idjornada, idvisitante, idvisitado}

  { hora, data, local } → {idjogo, golosVisitante, golosVisitado, ptsVisitante, ptsVisitado, idjornada, idvisitante, idvisitado}

  { idvisitante, idvisitado } → {idjogo, hora, data, local, ptsVisitante, ptsVisitado, golosVisitante, golosVisitado, idjornada}}

  Formas: BCNF: sim

  3NF: sim

- Equipa (idequipa , nome, sigla)

  FDs: { {idequipa} → {nome, sigla}

  {nome} → {idequipa, sigla}

  {sigla} → {idequipa, nome}}

  Formas: BCNF: sim

  3NF: sim

- Resultados (idresultados, pontos, ordem, idjornada→Jornada, idequipa→Equipa)

FDs: { {idresultados} → { pontos, idjornada, idequipa, ordem}

{idjornada,idequipa} → {idresultados, pontos, ordem}}

Formas: BCNF: sim

3NF: sim

- Jogador (idjogador, nome, idade, nacionalidade, número, posição, idequipa→Equipa)

  FDs: { {idjogador} → {nome, idade, nacionalidade, número, posição, idequipa}

  {nome, número, idequipa} → {idjogador, idade, nacionalidade, posição}}

  Formas: BCNF: sim

  3NF: sim

- Participa (idjogador→Jogador, idjogo → Jogo, marcador)

  FDs: { {idjogador, idjogo} → {marcador}}

  Formas: BCNF: sim

  3NF: sim

Observando as dependências funcionais apresentadas, vemos que todas as relações da base de dados seguem tanto a Forma Normal de Boyce-Codd (BCNF) como a 3ª Forma Normal. Todas as relações respeitam as condições de cada uma destas formas:

Uma relação está em **BCNF** se, para todo A → B não trivial, A é uma superkey/key;

Uma relação está em **3NF** se, para todo A → B não trivial, A é uma superkey/key OU B consiste apenas em atributos primos (atributos que são membros de pelo menos uma chave da relação).

Uma vez que, em todas as relações da base de dados, a parte esquerda de cada FD (conjunto de atributos A) determina funcionalmente todos os atributos da relação, conclui-se que A é (super)key. E assim, justifica-se a afirmação de que todas as relações estão em BCNF e 3NF.

Após a criação do modelo conceptual, bem como do modelo relacional, procuramos observar os nossos esquemas, para podermos determinar que restrições eram necessárias definir.

RESTRIÇÕES

Como nos é pedido, listamos aqui uma explicação de todas as **restrições** necessárias para a correta manutenção da nossa BD.

- Nenhuma instância de id (resultados, jogo, jornada, equipa visitante, equipa visitada, equipa, jogador) pode ser null, pelo que adicionamos a todas elas uma restrição NOT NULL;
- Na tabela RESULTADOS:
    - o idresultados é Primary Key, pois é a ele que nos referimos quando pretendemos consultar/apontar um determinado resultado. É por isso também Unique, ou seja, na tabela de resultados, não pode haver dois resultados com o mesmo id.
    - cada Resultado é referente a uma (e uma só) equipa e a uma (e uma só) jornada, pelo que idequipa e idjornada são Foreign Keys para, precisamente, uma Equipa e uma determinada Jornada.
    - o par (idequipa, idjornada) é Unique, visto que existe apenas um Resultado para cada equipa por cada jornada.
    - os pontos possuem a condição de serem maiores ou iguais a zero, pelo que acrescentámos CHECK de pontos>=0.

- Na tabela JOGO:
    - o idjogo é Primary Key, pois é a ele que nos referimos quando pretendemos consultar/apontar um determinado jogo. É por isso também Unique, ou seja, na tabela de jogos, não pode haver dois jogos com o mesmo id.
    - cada Jogo tem uma Equipa Visitante (equipa que “joga fora”), uma Equipa Visitada (equipa que “joga em casa”), pelo que idvisitante e idvisitado são Foreign Keys para as duas Equipas que disputam a partida;
    - cada Jogo está também inserido numa jornada, pelo que idjornada é Foreign Key que referencia a jornada em que decorre o jogo.
    - ainda, cada equipa apenas disputa um jogo por jornada. Assim, o par (idjornada, idvisitante), tal como o par (idjornada, idvisitado) têm uma restrição Unique.

    - para os golosVisitado e golosVisitante colocamos a condição >=0 e para os ptsVisitado e ptsVisitante, a condição <=3 e >=0).

- Na tabela EQUIPA:
    - o idequipa é Primary Key, pois é a ele que nos referimos quando pretendemos consultar/apontar uma determinada equipa. É por isso também Unique, ou seja, na tabela de equipas, não pode haver duas equipas com o mesmo id.

- Na tabela JOGADOR:
    - o idjogador é Primary Key, pois é a ele que nos referimos quando pretendemos consultar/apontar um determinado jogador. É por isso também Unique, ou seja, na tabela de jogadores, não pode haver dois jogadores com o mesmo id.
    - cada jogador pertence a uma Equipa, pelo que idequipa é Foreign Key que referencia a equipa a que o jogador pertence.
    - cada jogador pertence apenas a uma equipa, logo o par (idjogador, idequipa) deve ser único (restrição UNIQUE).
    - a idade deve ser >=0, pelo que colocámos essa condição com um CHECK.
    - o numero deve ser <=99 e >=0, pelo que colocámos essa condição com um CHECK.
    - a posição pode ser: 'goalkeeper', 'centre back', 'line player', 'left back', 'right wing', 'left wing'.

- Na tabela JORNADA:
    - o idjornada é Primary Key, pois é a ele que nos referimos quando pretendemos consultar/apontar uma determinada jornada. É por isso também Unique, ou seja, na tabela de jornadas, não pode haver duas jornadas com o mesmo id;
    - apenas existem 30 jornadas, logo incluímos uma restrição Check que apenas permite números de jornadas da ordem dos]0,30]. Assim, o numero possui a condição de ser >0 e <=30, pelo que colocamos um CHECK para a mesma.

- Na tabela PARTICIPA:
    - o par (idjogo, idjogador) é Primary Key, pois é esse par que demonstra qual jogador participa em qual jogo. É também Unique, dado que cada jogador apenas participa uma vez em cada jogo.
    - cada Participação é de um jogador, logo idjogador é Foreign Key que indica o Jogador ao qual nos referenciamos;
    - cada Participação é referente a um jogo, pelo que idjogo é Foreign Key que referencia, precisamente o jogo o jogador participa.
    - marcador apenas terá o valor de 0, no caso do jogador em questão não tiver marcado golo no jogo e 1, caso contrário.

INTERROGAÇÕES

1- Sigla, nome e pontuação das equipas que vão às competições europeias no final da época.

2- Nome dos jogadores menores de idade com nome próprio começado por 'J'.

3- Nome e idade dos 10 jogadores que marcaram golos em mais jogos e o número de jogos em que o fizeram.

4- Sigla da equipa que ficou em primeiro lugar em cada jornada e respetiva média de idades.

5- Percentagem das equipas cuja sigla não é de 3 caracteres.

6- Nome, posição e nacionalidade dos jogadores que marcaram golos em todos os jogos em que participaram.

7- Sigla e totais de golos marcados e sofridos por cada equipa na época.

8- Jogo com mais golos, jogo com menos golos e respetivas equipas participantes, resultados e data. Média de golos por jogo da época.

9- Resultado(V, D, E) de cada equipa por cada jornada. Caso não tenha jogado nessa jornada, indicar '-'.

10- Diferença entre a percentagem de jogos em que os jogadores do FCP, em média, marcaram golo e a percentagem de jogos em que cada jogador da liga, em média, marcou golo.

GATILHOS

1- Verifica se, ao inserir um novo jogo, ele pertence a uma jornada anterior à última inserida.

1- Ao adicionar um novo jogo, adiciona dois registos novos na tabela Resultados referentes às equipas que participam nesse jogo e respectivos totais pontos à jornada do jogo inserido.

2- Verifica se, ao inserir novos registos na tabela Participa, o jogador referente pertence a umas das equipas envolvidas no jogo em questão.

3- Verifica se a ordem de remoção dos registos na tabela Jogo é a correta (ordem decrescente de jornadas e, caso isso se verifique, remove também os registos na tabela Resultados associados às equipas e jornada envolvidas nesse jogo, para além de remover os registos de participação dos jogadores nesse jogo.

NOTAS IMPORTANTES:

- Os triggers devem ser adicionados após correr o povoar.sql.
- Os triggers devem ser corridos pela ordem certa, para garantir que os *gatilhoN_verifica.sql* funcionam como entendido.
- Nos gatilhos 2 e 3 os ficheiros *verifica* contêm instruções cujo propósito é falhar, para demonstrar as restrições impostas pelos mesmos.
- O gatilho 1 está também inicializado no *criar.sql*, uma vez que ele é necessário para a inserção de registos na tabela Resultados. Contudo, isso foi tido em conta quando criamos o *gatilho1_adiciona.sql*, e o ficheiro pode ser corrido como para qualquer outro trigger.
- Para o bom funcionamento desta base de dados foi necessário, para além destes gatilhos, implementar outros dois. Estes apenas impõem restrições, nomeadamente verificar se os jogos são inseridos por uma ordem de jornada crescente (gatilho ValidaJornada) e outro que nos impede de alterar registos na tabela Jogo (gatilho UpdateJogo) pois isto causaria incoerências em outras tabelas. Estes encontram-se no *criar.sql*, e são de um interesse/complexidade inferior aos escolhidos para os ficheiros.

NOTAS ADICIONAIS:

Entrega I:

Como explicado anteriormente, a nossa tabela Resultados conta com campos calculados dinamicamente através de triggers e por isso, de acordo com o enunciado, não povoamos essa tabela (verificar ficheiro povoar.sql). Além disso, como ainda não abordamos essa parte da matéria, o grupo teve dúvidas sobre a correta criação da própria, pelo que nos foi aconselhado expormos o nosso raciocínio neste relatório.

O atributo 'pontos' da tabela Resultados calcula o acumular de pontos de uma determinada equipa ao longo das épocas. Para fazer isso, é necessário verificar a tabela Jogo e encontrar, para aquela jornada (jogo.idjornada = resultados.idjornada), se a equipa foi visitante ou visitada (jogo.idvisitante = resultados.idequipa or jogo.idvisitado = resultados.idequipa). A partir daí basta nos obter o número de pontos ganhos no

jogo dessa jornada (jogo.ptsVisitante / jogo.ptsVisitado , conforme se a equipa é visitante ou visitada) e somar aos pontos já acumulados.

O atributo 'ordem' surge como consequência do atributo pontos, na medida em que comparamos a pontuação das várias equipas e determinamos qual se encontra em primeiro lugar, segundo, …

Por fim, gostaríamos de acrescentar que não fomos capazes de povoar a nossa Base de Dados com todos os dados da época. Nomeadamente, não incluímos todos os jogos, dado que, nos vários sites de consulta, por vezes encontrávamos informação conflituosa ou omitida. Por esse motivo, é possível que as multiplicidades expostas no nosso diagrama UML não estejam em conformidade com os dados inseridos. Contudo, é de referir que consultamos o professor Gabriel David, que encorajou esta decisão, uma vez que as informações presentes na BD continuam a ser pertinentes.

Entrega II:

Para a submissão da segunda entrega, foram efetuadas alterações nos ficheiros criar.sql e povoar.sql, sendo que, para a avaliação das queries e dos triggers é essencial a utilização das versões atualizadas dos ficheiros mencionados.

Criamos, também, duas tabelas auxiliares - PTSVISITANTE, PTSVISITADO - que apenas servem para guardar dados que serão utilizados num dos triggers. Foi necessária esta abordagem uma vez que o SQLite não suporta variáveis.

Avaliação da participação dos vários elementos do grupo:

Entrega I:

Todos os elementos do grupo contribuíram para a correta elaboração do projeto. O diagrama UML foi feito e revisto em conjunto, para que todos estivessem familiarizados com o tema (tarefa A).

Nas seguintes tarefas, dividimos o trabalho igualmente pelos membros. A Sofia Sá e o Guilherme Monteiro ficaram com a elaboração do Modelo Relacional, bem como com a Análise de Dependências Funcionais e Formas Normais (tarefas B e C). A Carolina Viana ficou encarregue da criação da BD em SQLite e consequente povoamento (tarefas D e F). Como a tarefa de povoamento foi mais trabalhosa do que o previsto

inicialmente, todos os membros do grupo acabaram por contribuir. A Carolina também adicionou as restrições à BD (tarefa E).

O relatório foi sendo elaborado por todos à medida que as tarefas eram concluídas.

Entrega II:

A Carolina Viana efetuou alterações no ficheiro povoar.sql enquanto que a Sofia Sá e o Guilherme Monteiro o fizeram no criar.sql.

A Sofia Sá alterou o UML e fez as respectivas mudanças consequentes no relatório final. Criou as queries (Tarefa G), com colaboração do Guilherme Monteiro e da Carolina Viana, que sugeriram e enriqueceram as mesmas. O Guilherme criou os Triggers (Tarefa H).

O projeto foi bem distribuído pelos 3 membros do grupo para uma boa fluidez de trabalho e máxima produtividade.