Instituto Politécnico de Viseu

Escola Superior de Tecnologia e Gestão de Viseu

Departamento de informática



Relatório do Trabalho Prático de Base de Dados II

Licenciatura em Engenharia Informática

Bruno Lopes nº17740

Carlos Costa nº17741

João Henriques nº 17751

Instituto Politécnico de Viseu

Escola Superior de Tecnologia e Gestão de Viseu

Departamento de informática

Relatório do Trabalho Prático de Base de Dados II

3º ano

1º semestre

Ano letivo 2020/2021

Bruno Lopes nº17740

Carlos Costa nº17741

João Henriques nº17751

**Agradecimentos**

Gostaríamos de agradecer aos professores da unidade curricular de Base de Dados II Paulo Tomé, Pedro Martins, João Neves, João Henriques e Artur Sousa por todo o conhecimento transmitido e por todo o auxílio e ajuda que nos foi facultada nas aulas e na realização do trabalho.

Índice

[1. Introdução 1](#_Toc64300698)

[2. Diagrama ER 2](#_Toc64300699)

[3. Base de Dados 4](#_Toc64300700)

[4. Aplicação Web 13](#_Toc64300701)

[5. Conclusão 19](#_Toc64300702)

[6. Bibliografia 20](#_Toc64300703)

**Índice de figuras**

[Figura 2‑1 - Modelo ER 3](#_Toc64300704)

[Figura 3‑1 – Declaração do cursor mais ações disciplinares 5](#_Toc64300705)

[Figura 3‑2 - Função calcular golos equipa 5](#_Toc64300706)

[Figura 3‑3 - Função calcular golos jogador 6](#_Toc64300707)

[Figura 3‑4 - Função número minutos de jogador num campeonato 7](#_Toc64300708)

[Figura 3‑5 - Função calcular golos de uma equipa num jogo 7](#_Toc64300709)

[Figura 3‑6 - Resultado da execução da view 8](#_Toc64300710)

[Figura 3‑7 - Estrutura XML 9](#_Toc64300711)

[Figura 3‑8 - Transação read commiter 10](#_Toc64300712)

[Figura 3‑9 - Resultado transação 10](#_Toc64300713)

[Figura 3‑10 - transações non-repeatable reads 11](#_Toc64300714)

[Figura 3‑11 - Resultado da transação 11](#_Toc64300715)

[Figura 3‑12 - Transações lost update 11](#_Toc64300716)

[Figura 3‑13 - Erro 11](#_Toc64300717)

[Figura 3‑14 - Resultado da transação 12](#_Toc64300718)

[Figura 4‑1 - View 13](#_Toc64300719)

[Figura 4‑2 - Criação da query select no método GET 15](#_Toc64300720)

[Figura 4‑3 - estruturação dos dados do formulário 15](#_Toc64300721)

[Figura 4‑4 - seleção do formulário a se usar 16](#_Toc64300722)

[Figura 4‑5 - criação da query update 17](#_Toc64300723)

[Figura 4‑6 - redirecionamento do utilizador para a página list 17](#_Toc64300724)

# Introdução

Este trabalho prático consiste no desenvolvimento de uma aplicação web de gestão da atividade de clubes. Um grupo desportivo poderá ter várias equipas em diferentes modalidades consoante o seu escalão e género. Uma equipa possui vários jogadores e estes podem mudar de equipa ou modalidade de uma época para a outra. Num campeonato existem vários jogos entre várias equipas, e nesses jogos serão contabilizadas as pontuações de cada equipa bem como as ações disciplinares aplicadas aos jogadores.

Para podermos desenvolver a aplicação web, tivemos de criar inicialmente o modelo ER da base de dados e depois implementá-la. Implementamos a Base de dados em Postgres utilizando o PgAdmin4 pois foi a linguagem que aprendemos no decorrer desta unidade curricular. Para criar o modelo ER utilizámos o PowerDesigner e em relação à aplicação web, esta foi desenvolvida em Django e Python através do Visual Studio Code.

.

# Diagrama ER

Os diagramas entidade relacionamento (ER) são um tipo de fluxograma que ilustra como as entidades se relacionam entre si dentro de um sistema. São mais utilizados para projetar base de dados relacionais.

Neste diagrama criámos as entidades Campeonatos, Modalidades, Generos, Epocas, Equipas, Clube, Faixa\_etaria, Jogos, jogadores, substituições, Tipo\_pontuacao, Acoes\_disciplinares, Pontuacoes e Tipo\_acao\_disciplinar. Criámos também três entidades que servem para os relacionamentos ternários. São elas pontuacoes\_jogadores\_jogos, jogadores\_jogos\_equipas e jogos\_jogadores\_acoes\_disciplinares. Tal como dito na introdução, um clube pode ter várias equipas e cada equipa pode ter várias modalidades consoante a faixa etária e o género. Ao longo do tempo um jogador pode jogar em várias equipas/modalidades. São também contabilizadas as pontuações dos jogos bem como as ações disciplinares ocorrentes nos jogos.

Para tudo isto, criámos o seguinte diagrama lógico com as entidades acima referidas bem como a sua interligação.

A Figura 2‑1 mostra o modelo ER que efetuámos.

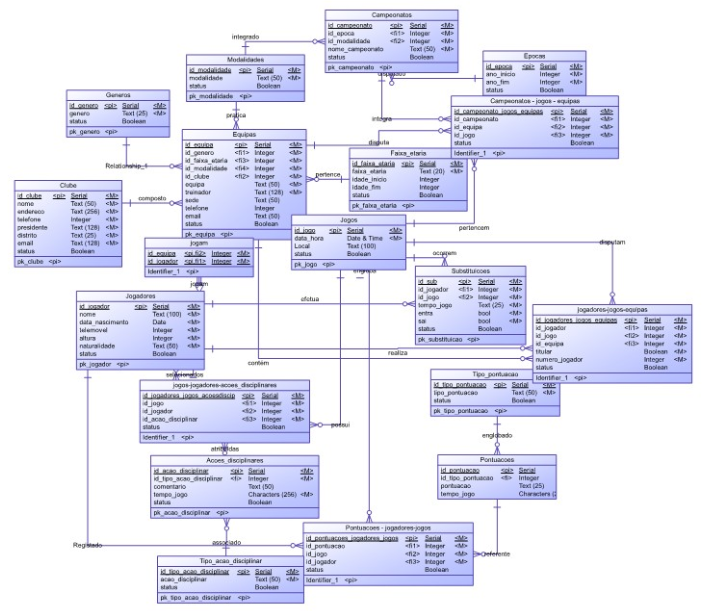


Figura ‑ - Modelo ER

# Base de Dados

A base de dados foi criada de acordo com o modelo ER acima referido. Esta foi criada em Postgres utilizando o PgAdmin. Operações DDL, Data definition language, são operações que lidam com esquemas e descrições da base de dados. Estas operações são do tipo Create, Alter, drop, etc. Já as operações DML lidam com a manipulação dos dados e incluem comandos SQL tais como Insert, Select, Update, Delete, etc. Estas operações servem para armazenar, modificar, apagar ou atualizar a base de dados. Estes dois tipos de operações foram executados e efetuados do lado da base de dados para que seja mantida a integridade dos dados.

Para além da criação da base de dados, criámos também dois procedimentos, quatro funções, um triggers, uma sequência, uma vista e xml. Um dos procedimentos consistia em encontrar os 2 jogadores com menos ações disciplinares e os 2 com mais ações disciplinares da época anterior e da época corrente respetivamente num determinado jogo entre duas equipas. Os jogadores com mais ações disciplinares serão substituídos pelos jogadores com menos ações disciplinares.

Para pesquisarmos os jogadores com menos ações disciplinares criámos um cursor com o id do campeonato, o id da equipa e o id do jogo a serem passados por parâmetro. A figura seguinte mostra a declaração do cursor que foi efetuado para determinar os dois jogadores com mais ações disciplinares.

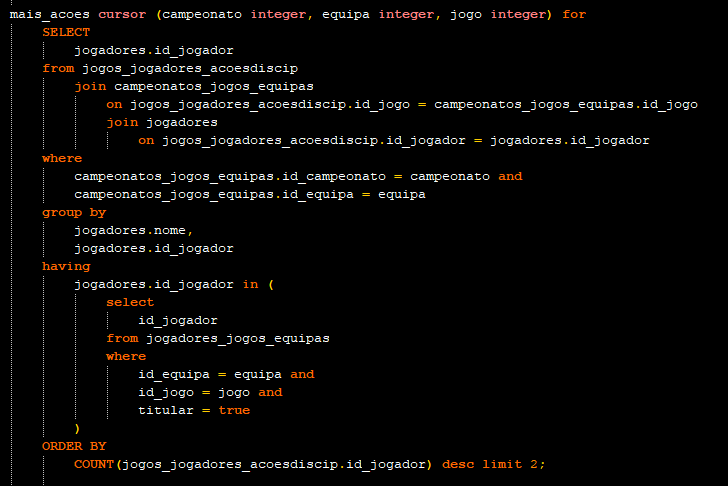


Figura ‑ – Declaração do cursor mais ações disciplinares

Após a declaração de ambos os cursores, estes serão abertos e serão verificados quais os jogadores a sair e quais os jogadores a entrar. Seguidamente, procedemos à substituição dos jogadores através do comando Insert.

**FALAR SEGUNDO PROCEDIMENTO**

Em relação às funções, criámos uma para calcular o total de golos marcado por uma equipa num determinado ano e outra que calcula o total de golos marcados por um jogador num determinado ano. Para a primeira função, foram passados por parâmetro o id da equipa e o id do campeonato. Será retornado a contagem do número de pontuações efetuadas por essa equipa nesse ano. A Figura 3‑2 mostra a função criada.

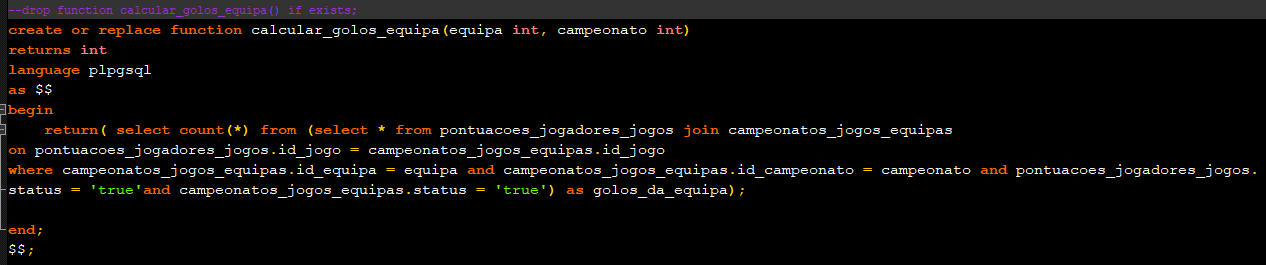


Figura ‑ - Função calcular golos equipa

A função para calcular os golos do jogador é estruturalmente semelhante à função referida anteriormente com a diferença de ser para o jogador. A Figura 3‑3 mostra a função realizada

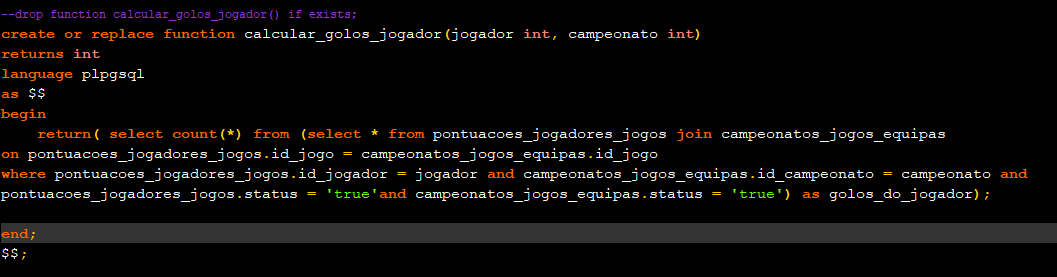


Figura ‑ - Função calcular golos jogador

Criámos uma função que permite calcular os minutos de jogo que um determinado jogador teve num determinado campeonato. Estas duas variáveis mencionadas foram passadas como parâmetro na função. Um jogador pode ser titular e jogar o jogo todo, ser titular e ser substituído ou então ser substituto e entrar numa determinada altura. Deste modo, criámos uma variável que irá guardar o número do minuto que um jogador efetuou quando era substituto e entrou durante o jogo. Seguidamente criámos outra variável que irá subtrair o minuto em que o jogador entrou ao tempo que o jogo demora. Criámos outra variável que guarda o número de minutos em que o jogador foi titular e foi substituído. Por fim, criámos uma variável que guarda o número de minutos que um jogador foi titular sem ser substituído. Para tal, primeiramente é feita a contagem dos jogos que este jogador foi titular sem ser substituído e após isso é multiplicado pelo tempo total do jogo. No fim, Somamos todas estas variáveis numa só e retornamo-la na função. A figura seguinte mostra a estrutura efetuada nesta função.

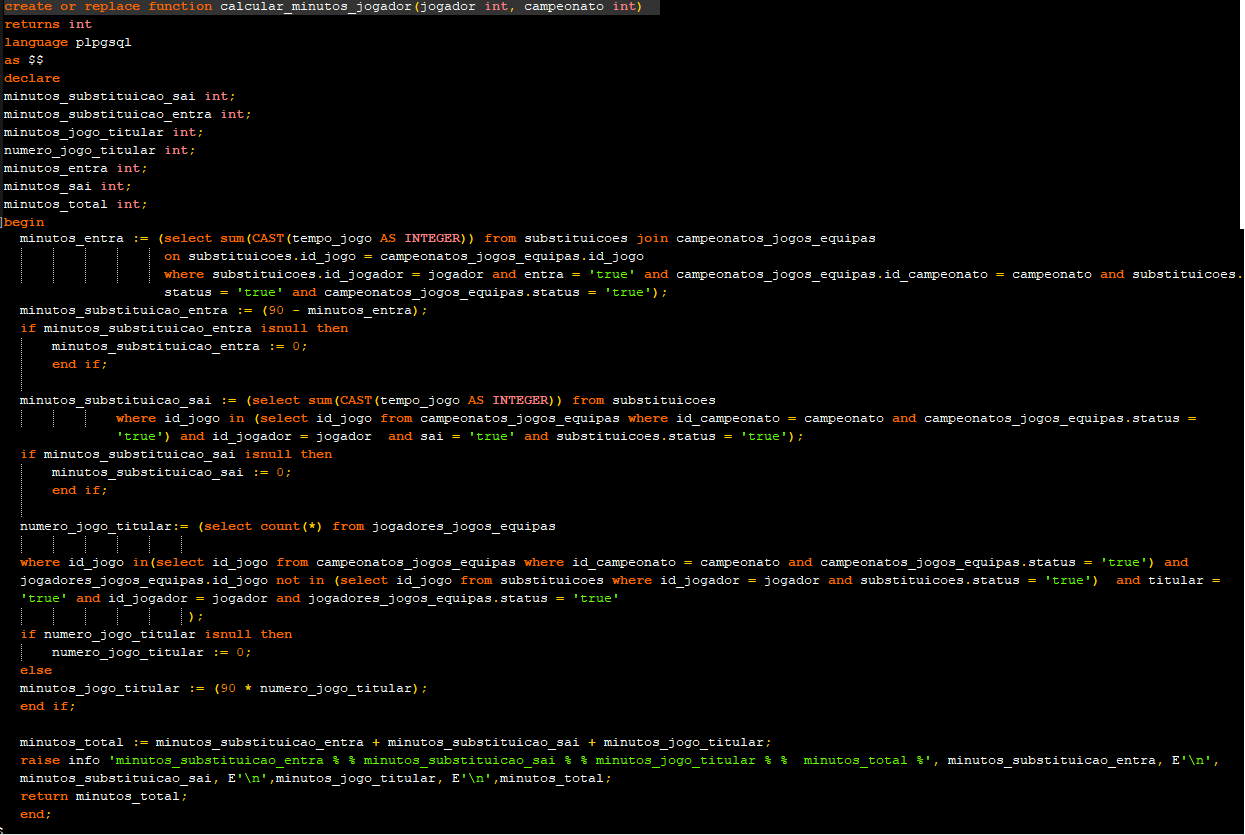


Figura ‑ - Função número minutos de jogador num campeonato

A última função que criámos serve para calcular o número de golos de uma equipa num determinado jogo. Isto é feito através de um count da tabela pontuações\_jogadores\_jogos onde o jogo é igual ao id de jogo passado por parâmetro e também o id de jogo no qual aquela equipa pertence. A Figura 3‑5 mostra a função descrita acima.

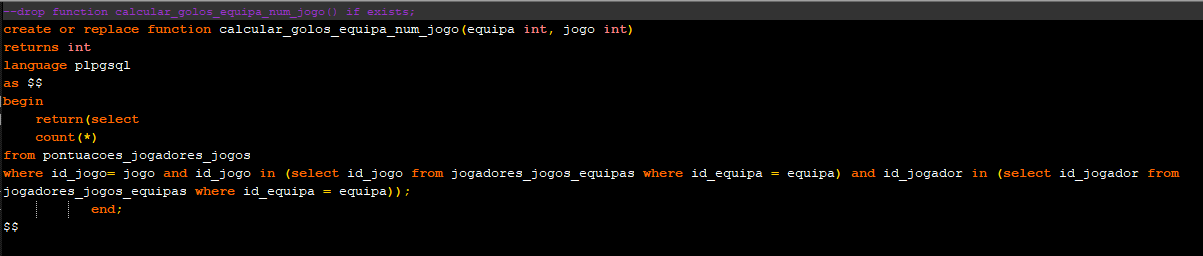


Figura ‑ - Função calcular golos de uma equipa num jogo

O triggers que criámos serve para manter sempre a base de dados atualizada. Sempre que é feito um update, delete ou insert é inserido essa informação numa tabela, servindo isto como uma espécie de Logs.

A sequencia pretendida foi criada na coluna comentário da tabela acoes\_disciplinares. Será criada uma sequência numérica que irá ser agregada à palavra “acaodisciplinar”. O resultado disto será por exemplo “acaodisciplinar1”, “acaodisciplinar2”, etc.

Em relação às vistas, criámos uma vista que mostra o resumo dos jogos das equipas. Nesta vista será mostra a data do jogo, o campeonato, as equipas e o resultado do jogo bem como o jogador com mais golos e mais penalizado de cada equipa. A seguinte figura mostra a execução da vista.



Figura ‑ - Resultado da execução da view

Como a base de dados está estruturada de forma relacional, criámos uma estrutura xml para mostrar os dados dos jogadores para o último campeonato. Para o xml, criámos um xmlelement chamado jogadores e dentro desta teremos a agregação dos dados pessoais do jogador, bem como a equipa e o clube a que este pertence. Possuímos ainda o número de golos e número de ações disciplinares para completar assim a informação dos jogadores.

A figura seguinte mostra a estrutura XML criada.



Figura ‑ - Estrutura XML

No controlo de transações, devido ao facto do nível de isolamento por defeito no postgreSQL ser “read committer”, não é possível obter “dirty reads[[1]](#footnote-1)” nem “phantom reads[[2]](#footnote-2)”.

Para podermos verificar isso utilizámos duas transações em simultâneo. Na primeira transação fizemos um insert numa tabela, de seguida selecionámos em ambas as transações todos os valores da tabela e verificámos que apenas na primeira transação foi possível ver o novo insert.

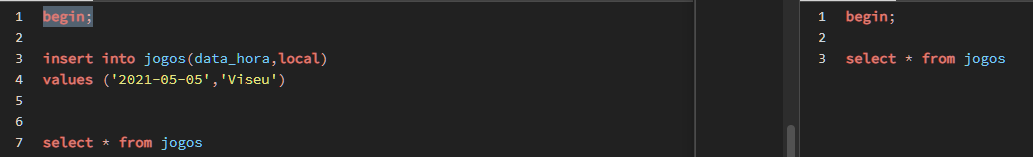


Figura ‑ - Transação read commiter



Figura ‑ - Resultado transação

Outro problema nas transações são as “non-repeatable reads”. Estas leituras ocorrem quando numa transação é lida uma linha e depois ao lê-la novamente, o resultado é diferente. Isto acontece devido a outra transação ter feito um commit que alterou essa linha.

De modo a evitar este problema de “non-repeatable reads” o postgreSQL permite alterar o nível de isolamento da transação como “isolation level repeatable read”. Deste modo, é garantido que qualquer leitura irá retornar sempre o mesmo resultado que a primeira leitura.

Para verificarmos esta situação, utilizámos duas transações em simultâneo, na primeira com o nível de isolamento “isolation level repeatable read” e na segunda efetuámos a alteração da linha. De seguida, verificámos que os valores da tabela na primeira transação continuaram os iguais.

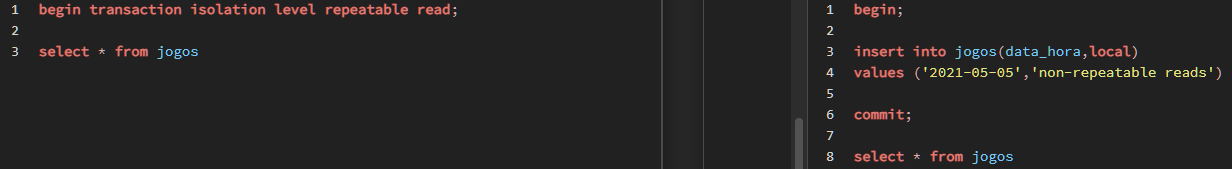


Figura ‑ - transações non-repeatable reads



Figura ‑ - Resultado da transação

Por último, temos os “lost updates” que acontecem quando uma alteração feita numa transação é perdida ou substituída por outra alteração feita noutra transação. Para evitar este problema, em semelhança ao este problema de “non-repeatable reads”, o postgreSQL permite alterar o nível de isolamento para “isolation level serializable” que vai prevenir este problema de acontecer, impedindo que sejam feitas alterações, noutras transações, na mesma linha que está a ser editada na primeira transação.

Para testarmos este nível de isolamento, na primeira transação atualizámos uma linha de uma tabela e na segunda transação, utilizando o “isolation level serializable”, tentámos alterar a mesma linha. Ao fazer o update na segunda transação, a transação bloqueou e ficou á “espera” de que a primeira transação fosse executada e após fazermos o commit na primeira transação, a segunda transação foi abortada. No final, podemos observar que a linha foi alterada de acordo com a primeira transação.

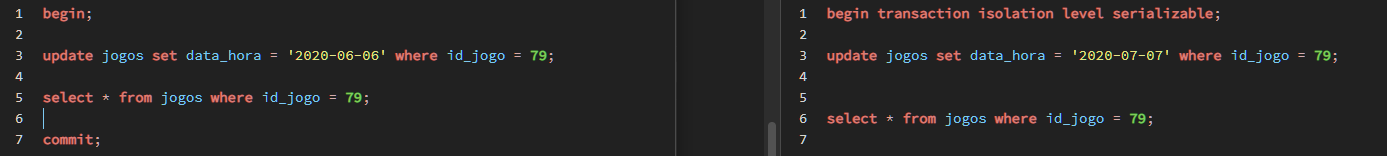


Figura ‑ - Transações lost update

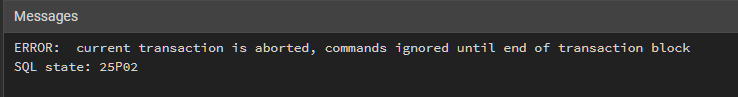


Figura ‑ - Erro

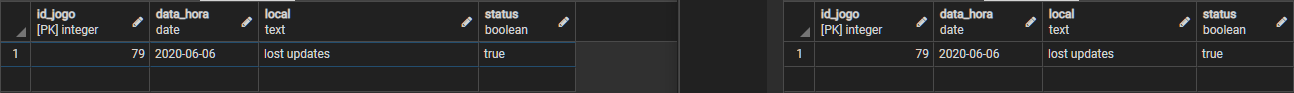


Figura ‑ - Resultado da transação

# Aplicação Web

A aplicação web consiste na visualização dos dados bem como na criação, atualização e eliminação de dados.

Começámos por criar os urls e importámos os models da base de dados para o django. Posteriormente criámos as classes para todas as tabelas no ficheiro forms.py e após criadas as classes, definimos as views para cada uma das tabelas.

Na parte da criação dos dados, criámos uma página html para cada uma das tabelas. Através da view irá apresentar primeiramente a página do formulário e após clicarmos no botão de submissão irá verificar se o formulário é válido. Se for, ele irá guardar os dados na tabela correspondente na base de dados. A figura seguinte mostra a view efetuada para a tabela dos tipos de pontuação.

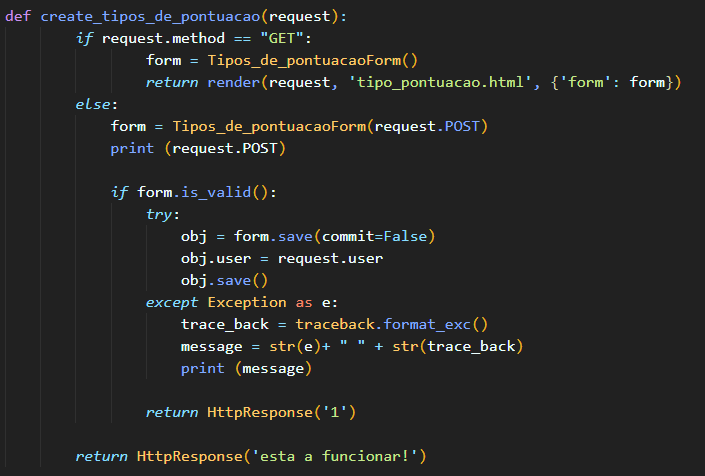
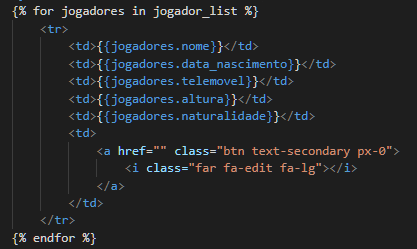


Figura ‑ - View

Para listarmos as tabelas da base de dados, fizemos o import das tabelas presentes nos models para as views. De seguida, utilizámos a função presente na figura seguinte para fazer o render dos dados de todas as tabelas e de uma view para a página “list.html”.



Na página “list.html”, percorremos todos os dados de cada tabela e fizemos a sua listagem.



Para se atualizar os dados foi usado a generalização onde temos, no caso de ser um request do tipo GET, a criação de uma query do tipo select de forma a obtermos os dados a editar da base de dados para posteriormente se criar o formulário de preenchimento correspondente à tabela pretendida, tal como se pode ver nas figuras seguintes.

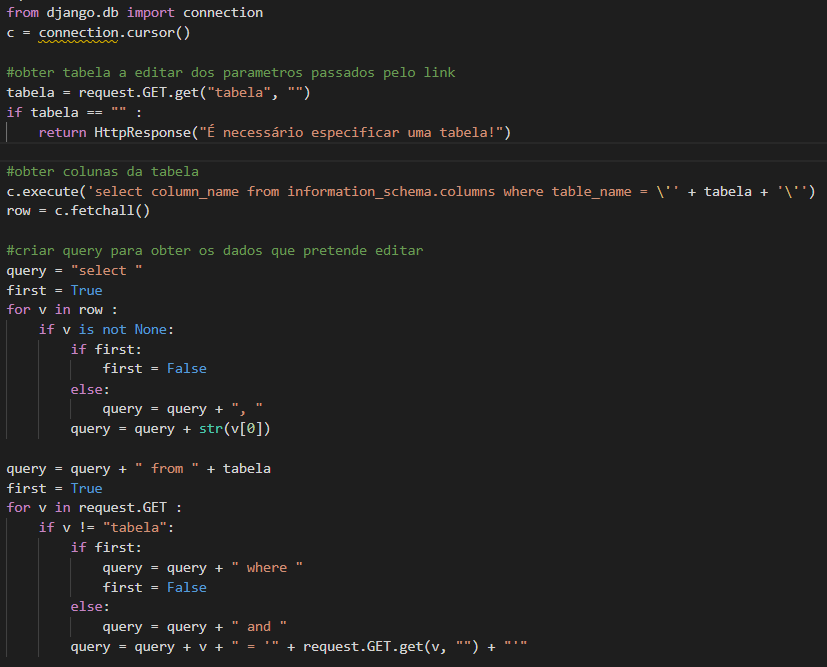


Figura ‑ - Criação da query select no método GET

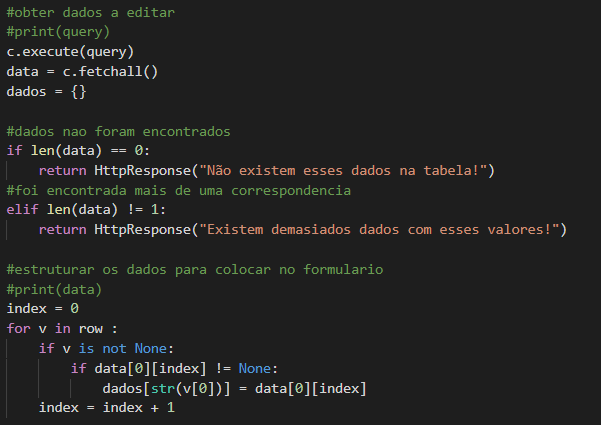


Figura ‑ - estruturação dos dados do formulário

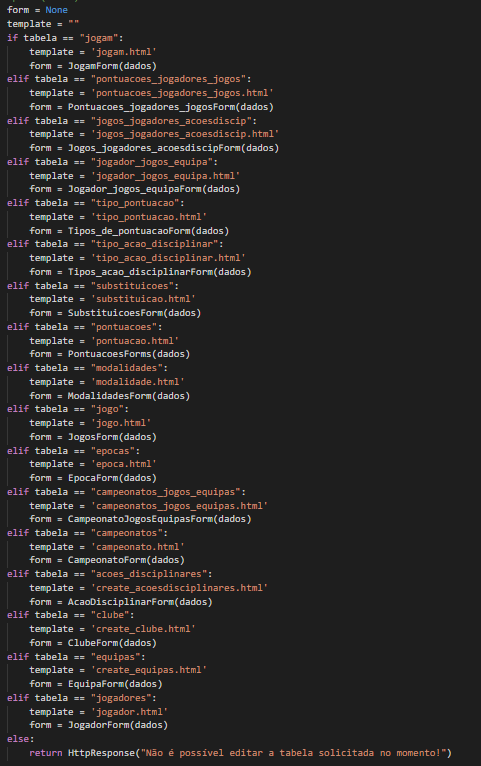


Figura ‑ - seleção do formulário a se usar

Por fim, para se poder atualizar os dados através do botão Submit do formulário criado anteriormente, caso o request seja do tipo POST, são usados os parâmetros passados pelo link no pedido anterior de forma a se identificar a tabela correspondente ao formulário e ,posteriormente, criar a query update que vai atualizar os dados do objeto selecionado, como se pode ver na figura seguinte.



Figura ‑ - criação da query update

O utilizador é posteriormente redirecionado para a página anterior que corresponde ao caminho ‘/list’.



Figura ‑ - redirecionamento do utilizador para a página list

Em relação à eliminação de dados, esta é procedida de forma semelhante à atualização. Possuimos em cada tabela uma coluna status em que o valor é um boolean. Caso o valor seja True, esses dados serão mostrados e caso seja False, os dados dessa linha não serão mostrados porque é como se estivessem eliminados. Em base de Dados é de bom grado não eliminar definitivamente os dados pois pode causar erros e complicações na base de dados. Para eliminar os dados basta fazer um update do status dessa linha de dados para False.

# Conclusão

Em virtude da realização do trabalho prático, podemos concluir que foram aplicados todos os conhecimentos adquiridos durante as aulas, através dos docentes da disciplina. Como resultado, obtivemos a configuração de um servidor DHCP, DNS, web e base de dados.

Este trabalho foi muito importante para a compreensão do funcionamento do protocolo DHCP.

# Bibliografia

Alves, Gustavo Furtado de Oliveira. «Você precisa saber o que é SQL!» { Dicas de Programação }, 26 de Abril de 2013. https://dicasdeprogramacao.com.br/o-que-e-sql.

«O que são DDL e DML?» Acedido 15 de Fevereiro de 2021. https://qastack.com.br/programming/2578194/what-are-ddl-and-dml.

«PowerDesigner». Em *Wikipédia, a enciclopédia livre*, 15 de Maio de 2019. https://pt.wikipedia.org/w/index.php?title=PowerDesigner&oldid=55145925.

DevMedia. «Qual a Diferença Entre DDL e DML?» Acedido 15 de Fevereiro de 2021. https://www.devmedia.com.br/forum/qual-a-diferenca-entre-ddl-e-dml/563525.

«Transaction Isolation in PostgreSQL - pgDash». Acedido 15 de Fevereiro de 2021. https://pgdash.io/blog/postgres-transactions.html.

1. Dirty reads – ex: Temos duas transações inacabadas e numa delas conseguimos ver as alterações da outra. [↑](#footnote-ref-1)
2. Phantom reads – ex: Quando temos uma dirty read e na transação onde foram feitas as alterações é dado o rollback. Na segunda transação ainda será possível ver as alterações feitas na transação. [↑](#footnote-ref-2)