Instituto Superior Técnico

Sistemas de Informação e Base de Dados

2021/2022

SIBD Project - Part 3

Professor:
Bruno Martins

Alunos:
Diogo Martins, 93047
Miguel Eugénio, 93308
João Rocheteau, 102247

30 de Janeiro de 2022

Grupo 25

Turno 2º feira - 10:00/11:30 e 3º feira - 11:00/12:30

Contribuição dos autores:

- Diogo Martins - 35% - 10h
- Miguel Eugénio - 30% - 9.5h
- João Rocheteau - 35% - 10h

1 Arquitetura da aplicação

No exercício 5 do laboratório é-nos pedida uma aplicação que permita ao utilizador realizar várias tarefas como, remover e adicionar *owners*, criar e remover reservas, etc. A aplicação foi desenvolvida utilizando *Python CGI scripts* e *HTML pages*, tendo também em atenção a proteção contra *SQL Injection*.

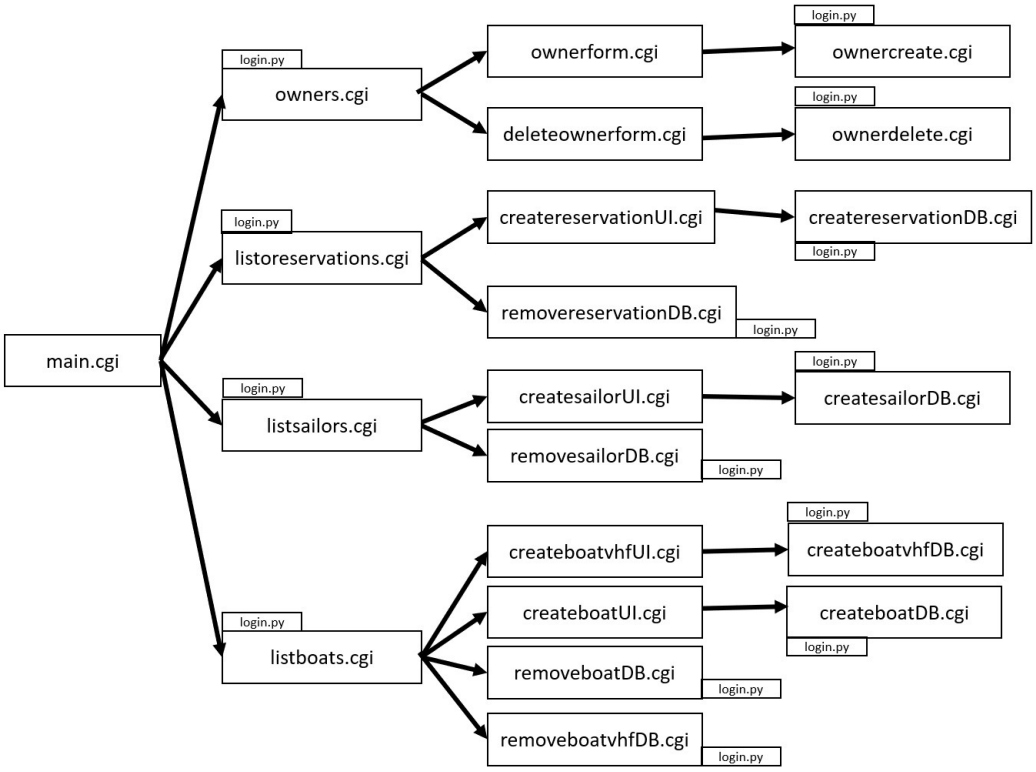

Figura 1: Diagrama da arquitetura da aplicação

Na imagem anterior (Fig. 1) é apresentado um diagrama com a estrutura da aplicação, que vamos passar a explicar em detalhe. De uma forma geral a aplicação dá a possibilidade ao utilizador de escolher a tarefa que pretende executar, através de quatro "índices"(um para cada exercício proposto), que o direciona para a(s) tabela(s) que contêm os dados associados à tarefa pretendida. Em seguida, este pode escolher o que quer fazer, remover ou adicionar, e depois executar essa tarefa.

A nossa aplicação começa com um *main.cgi*, um ficheiro que permite aos utilizadores aceder à lista de tarefas que pretendem realizar, tal como pedido. De referir também que a hiper-ligação para boat, possui na sua interface a possibilidade de remover / criar não só boats como também boats vhf.

Existem 2 operações que os utilizadores podem realizar: CREATE e DELETE. Começando com a operação DELETE, para operações onde a tabela afectada é *sailors*, *boat* ou *reservation*, os utilizadores, escolhendo uma linha da referida tabela, podem eliminá-la clicando apenas num botão "DELETE" na respetiva posição (X linhas de uma tabela vão ter X botões DELETE). Os valores dos parâmetros necessários à eliminação de uma determinada linha são enviadas através de um endpoint (e.g a chave da tabela) e são recuperados pelo ficheiro responsável por executar o comando DELETE (os ficheiros de mais baixo nível no diagrama).

Para a tabela *owner* existe só um botão DELETE, onde o utilizador tem que introduzir as variáveis que correspondem à linha que quer eliminar através de um form, no qual estas informações também serão recuperadas no ficheiro seguinte (de mais baixo nível) para proceder à execução. Aqui, existe um passo extra que é a introdução do valores pelo utilizador, onde antes não era necessário. Este processo será necessário nos CREATE's.

Para o CREATE, como é sabido, é necessário o utilizador colocar todas a informações referentes à nova linha a ser criada, tal como seria na base de dados, através de um INSERT. Existe um botão de "CREATE"para cada exercício que direciona os utilizadores para um FORM onde poderão introduzir as novas variáveis e, mais uma vez, serão posteriormente direcionados para os ficheiros que procedem a execução.

Se as *constraints* não forem quebradas, a aplicação consegue responder ao que foi pedido.

De maneira a combater os ataques através de *SQL Injection* foram usadas as técnicas expostas nas aulas teóricas e nos laboratórios, como podemos ver na imagem abaixo (Fig. 2). Isto força o criador do código a definir primeiro todo o código SQL e só depois verifica cada parâmetro que estará na *query*, isto permite à base de dados distinguir entre código e dados inseridos pelo utilizador, não permitindo assim que seja inserido qualquer tipo de código que seja prejudicial para a base de dados.

```
# Making query
sql = 'INSERT INTO boat_vhf(mmsi, cni, iso_code) VALUES (%s, %s, %s);'
data = (mmsi, cni, iso_code)
# The string has the {}, the variables inside format() will replace the {}
print('<p>{}</p>'.format(sql % data))
# Feed the data to the SQL query as follows to avoid SQL injection
cursor.execute(sql, data)
```

Figura 2: Proteção contra SQL Injection

O link para aceder à página é o seguinte:

https://web2.tecnico.ulisboa.pt/ist1102247/main.cgi

2 Indexes

Em relação ao índices temos dois exercícios diferentes com o objetivo de criá-los de maneira às *queries* terem um menor custo computacional.

2.1 Primeiro exercício

Neste caso temos uma *query* onde se juntam duas tabelas (boat e country) e se selecionam os dados impondo restrições a dois atributos (year e name de cada tabela, respetivamente), estes dois atributos não são *primary keys* logo não têm índices associados. Como estes dois atributos não estão na mesma tabela não podemos criar um índice composto, desse modo teremos de criar um índice em cada tabela.

Como temos uma *range-query* não poderemos usar um *hash index*, a melhor opção a usar será *B+Tree* pois é a mais eficiente neste tipo de *queries* e permite que os nós estejam o melhor distribuídos possível, possibilitando assim um menor custo computacional.

B+Tree é uma melhoria de *B-Tree*, esta é uma *self-balancing search tree*, quando temos um quantidade muito grande de dados é muito demoroso aceder ao disco, o objetivo é diminuir os número de acessos tornando assim todos os processos mais rápidos. Dividindo os dados em vários nós de tamanho semelhante por vários níveis é possível fazê-lo. Na *B+Tree* os dados podem ser guardados nos nós finais (leaf nodes) e estes estão conectados entre si, o que torna a pesquisa mais rápida que *B-Tree*. Na Fig. 3 podemos ver um exemplo da *B+Tree*.

O Index criado no primeiro exercicio é o apresentado na Fig., 4 serão os seguintes:

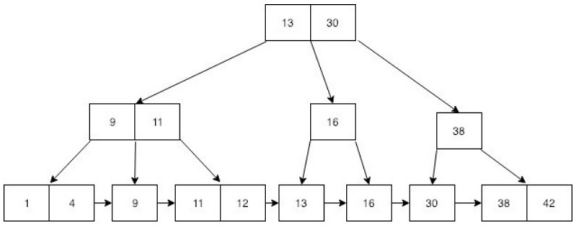

Figura 3: B+Tree

```
CREATE INDEX b_year_index ON boat (year);
CREATE INDEX c_name_index ON country (name);
```

Figura 4: Índice 1

Sem a criação dos índices para obter o resultado da *query*, o sistema iria passar por todos os dados da tabela um por um até encontrar quais os barcos que foram criados a partir de um certo ano e depois procurar novamente os barcos que correspondem a essa condição e que foram registados no país indicado. Utilizando os índices em cada um destes atributos a procura será muito mais rápida.

Estando os dados organizados pela árvore, a pesquisa será ordenada para o ramo da árvore que contem os anos perto destes indo descendo os níveis da árvore até chegar à condição pretendida. Este índice apenas será eficaz assumindo que os dados estão distribuídos equilibradamente e que a condição do ano imposta restringe os dados, ou seja, se procurarmos um barco registado depois de 1980 a maior parte destes correspondem a esta restrição, logo o índice não será útil (será uma melhor opção retirar todos os dados diretamente da tabela). O mesmo acontecerá posteriormente na pesquisa do nome do país, a pesquisa será direcionada logo para os nomes mais próximos (na árvore) do país escolhido, sendo cada vez mais afunilada ao longo da árvore até chegar à restrição pedida. Isto torna muito mais rápida a execução da *query*.

2.2 Segundo exercício

Neste exercício queremos contar quantas viagens partem de uma certa localização (cujo o nome deve conter as sequência de letras indicadas em LIKE '%some pattern%') entre duas datas de inicio dadas.

Assim como no exercício anterior sendo uma *range-query* na seleção das datas, não poderemos usar um *hash index*, logo será usado um índice com *B+Tree*. As datas serão ordenadas pela árvore como explicado anteriormente e assim a pesquisa será muito mais rápida, sendo procuradas as datas correspondestes à restrição imposta.

No entanto para o nome da localização um índice não trará um grande beneficio, visto que o utilizador apenas tem de especificar um padrão que o nome terá de conter, podendo este padrão encontrar-se dentro da *string*. Como os indicies organizam as *strings* pelo inicio, ao especificar um padrão no interior desta obrigará o sistema a procurar em todos os blocos ou ramos (dependendo do tipo de *index* usado) pelo padrão, logo não terá utilidade. Assim optamos apenas pelo o uso do índice na data de inicio como vemos na figura seguinte (Fig. 5).

```
CREATE INDEX t_start_date_index ON trip (start_date);
```

Figura 5: Índice 2