Projeto BD - Parte 3

Turno BDL13 – Grupo 103
Professor – Gonçalo Freire

Realizado por:

João Vaz – 98946
Miguel Pereira – 99109
Vasco Afonso – 99134

Percentagem de contribuição de cada aluno :

João Vaz – 33.3%
Miguel Pereira – 33.3%
Vasco Afonso – 33.3%

Esforço Total:

22 horas(cada aluno)

1. Bases de Dados

1.1. Esquema

- *Para inicializar a Base de Dados criamos tabelas e carregamos os registos necessários de cada tabela, seguindo o modelo relacional apresentado no **Anexo A**. Colocamos os tamanhos corretos para cada campo e associamos cada campo como sendo Foreign Key ou Primary Key, seguindo as especificações do Modelo Relacional. Adicionamos ainda uma constraint a cada combinação de keys com o objetivo de não existirem tabelas com valores duplicados.*

1.2. Carregamento

- *Para o carregamento gerámos, através de um script python, valores na tabela com as relações necessárias para mais tarde conseguir testar corretamente os problemas enunciados.*

2. Restrições de Integridade

(RI-1) Uma Categoria não pode estar contida em si própria

- *Para esta restrição, criámos um Trigger que ao adicionar uma nova relação de tem_outra verifica se o nome da super_categoria e da sub_categoria não são o mesmo.*

(RI-4) O número de unidades repostas num Evento de Reposição não pode exceder o número de unidades especificado no Planograma

- *Para esta restrição, criámos um Trigger que verifica se o número de unidades no evento_reposicao, a adicionar, não é maior que o número de unidades no seu planograma respetivo, através dos campos ean, nro, num_serie e fabricante.*

(RI-5) Um Produto só pode ser reposto numa Prateleira que apresente (pelo menos) uma das Categorias desse produto

- *Para esta restrição, criámos um Trigger que primeiramente verifica a categoria da prateleira a ser resposta, depois conta o número de relações tem_categoria, em que a categoria corresponde à categoria da prateleira e o ean ao ean do produto a ser reposto. Por fim, verifica se esse número é zero, neste caso o produto não pode ser reposto.*

3. SQL

Qual o nome do retalhista (ou retalhistas) responsáveis pela reposição do maior número de categorias?

- *Criação de uma tabela temporária " freq_cte" em que as colunas são o nome dos retalhistas e o número de entradas da tabela retalhista junta com a tabela responsavel_por(freq). De seguida selecionamos todos os retalhistas que tenham o valor na coluna freq maior.*

Qual o nome do ou dos retalhistas que são responsáveis por todas as categorias simples?

- *Seleciona os retalhistas associados apenas a categorias simples no responsavel_por e utiliza a última relação para manter apenas os que aparecem o mesmo número de vezes que o número de categorias simples.*

Quais os produtos (ean) que nunca foram repostos?

- *Selecionar todos os ean's da tabela produtos que não apareçam na tabela evento_reposição.*

Quais os produtos (ean) que foram repostos sempre pelo mesmo retalhista?

- *Seleção dos ean's de uma tabela temporária "ean_freq" que contém os ean's dos produtos que quando aparecem na tabela evento_reposicao estão sempre associados ao mesmo tin.*

4. Vistas

Criação da vista Vendas com os seguintes campos:

- *ean, através de evento_reposicao*
- *cat, através de tem_categoria*
- *ano, trimestre, mes, dia_mes e dia_semana, através de extract de evento_reposicao.instante*
- *distrito e concelho, através de ponto_de_retalho*
- *unidades, através de evento_reposicao*

5. Desenvolvimento da aplicação

a. Introdução

- *Com vista a interagir com a base de dados criada previamente, desenvolvemos um protótipo de aplicação web em scripts Python CGI e páginas HTML, com o propósito de solucionar os seguintes problemas:*
  - I. *Inserir e remover categoria e sub-categorias;*
  - II. *Inserir e remover um retalhista, com todos os seus produtos;*
  - III. *Listar todos os eventos de reposição de uma IVM, apresentando o número de unidades repostas por categoria de produto;*
  - IV. *Listar todas as sub-categorias de uma super-categoria, a todos os níveis de profundidade.*
- *Para isto criámos um ficheiro Python CGI principal, "app.cgi", e um html para cada página web, contida na pasta "templates".*
- *Link:* https://web2.ist.utl.pt/~ist199134/app.cgi/

b. Menu Principal

- *Começámos por desenvolver um menu principal onde cada item nos leva a uma página com as ações pretendidas, relacionadas com o mesmo, criando uma tabela no ficheiro index.html e renderizando-a em app.cgi*

```python
@app.route("/")
def main_menu():
    try:
        return render_template("index.html")
    except Exception as e:
        return str(e)  # Renders a page with the error.
```

1. Route "/" - app.cgi

Menu
Categorias
Retalhistas
Eventos de Reposição
Sub-categorias

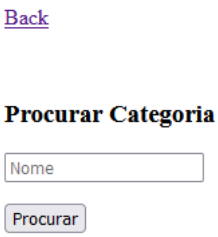
2. Tabela - Menu Principal

c. Categorias

- *A página de Categorias, em app.cgi, que renderiza cat.html, permite-nos consultar todas as categorias presentes no sistema bem como adicionar e remover. Optámos por fazer isto da seguinte forma:*
    - *Adicionar categoria pede ao utilizador o nome da categoria a adicionar e adiciona este à tabela categoria e categoria_simples, caso este ainda não exista;*
    - *Remover categoria aparece na tabela ao lado do nome da categoria a remover de forma a ser mais fácil de executar a ação;*
    - *Procurar a categoria escrevendo o seu nome, restringe a tabela à categoria com o nome procurado.*

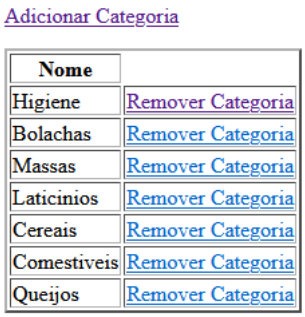
i. Adicionar

- *Ao entrar na opção "Adicionar categoria" é renderizado o ficheiro categorias.html com um form que regista o nome da nova categoria e executa a query em app.cgi, route "/updt_cat_add".*



ii. Remover

- *Ao clicar "Remover Categoria" executa a query em app.cgi, route"/updt_cat_remove", recebendo a categoria correspondente na tabela como argumento e por fim é redirecionado para a página inicial das categorias (route"/categoria")*



3. Route"/categoria" - app.cgi

iii. Procurar

- *Ao clicar "Procurar" é renderizado o ficheiro search.html, através dos parâmetros passados em app.cgi, route"/search_categoria", que representa os dados introduzidos no form e ao clicar procurar executa a query presente em app.cgi, route"/categoria" com os parâmetros passados e demonstra então uma tabela com a categoria procurada.*

d. Retalhistas

- *A página de Retalhistas, em app.cgi que renderiza retalhista.html, permite-nos consultar todos os Retalhistas presentes no sistema bem como adicionar e remover com todos os seus produtos. Optámos por fazer isto da seguinte forma:*
    - *Adicionar retalhista pede ao utilizador o tin e o nome do retalhista a adicionar e adiciona este à tabela retalhista, caso este ainda não exista;*
    - *Remover retalhista aparece na tabela ao lado do tin e do nome do retalhista a remover de forma a ser mais fácil de executar a ação;*
    - *Procurar um retalhista pode ser feito escrevendo o tin e restringe a tabela ao retalhista com o tin procurado.*

i. Adicionar

- *Ao entrar na opção "Adicionar Retalhista" é renderizado o ficheiro add_retalhista.html com um form que regista o nome da nova categoria e executa a query em app.cgi, route "/updt_retalhista".*

ii. Remover

- *Ao clicar "Remover Retalhista" executa a query em app.cgi, route"/remove_retalhista", recebendo o tin e nome do Retalhista correspondente na tabela como argumentos e por fim é redirecionado para a página inicial das categorias (route"/retalhista")*

4. Route"/retalhista" - app.cgi

iii. Procurar

- *Ao clicar "Procurar" é renderizado o ficheiro search.html, através dos parâmetros passados em app.cgi, route"/search_retalhista", que representa os dados introduzidos no form e ao clicar procurar executa a query presente em app.cgi, route"/retalhista" com os parâmetros passados e demonstra então uma tabela com o retalhista procurado.*

e. Eventos de Reposição

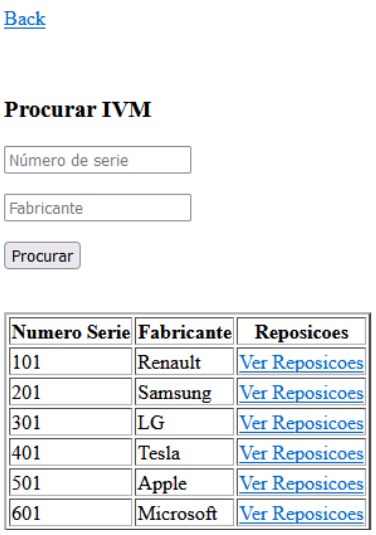
- *A página de Eventos de Reposição, em app.cgi que renderiza ivm.html, permite-nos consultar todas as Ivm's presentes no sistema bem como apresentar o número de unidades repostas por categoria de produto numa certa Ivm.*
  Optámos por fazer isto da seguinte forma:
  - *Ver reposições aparece na tabela ao lado do Número de Série e Fabricante a listar de forma a ser mais fácil de executar a ação;*
  - *Procurar uma IVM pode ser feito escrevendo o Número de série ou o Fabricante, restringindo a tabela às IVM com as características encontradas.*

i. Reposições

- *Ao entrar na opção "Ver Reposições" é renderizado o ficheiro show_repositions.html que nos apresenta uma tabela com duas colunas Categoria e Unidades. A tabela contém o número de unidades repostas por categoria de produto.*

ii. Procurar

- *Ao clicar "Procurar" é renderizado o ficheiro search.html, através dos parâmetros passados em app.cgi, número de série, fabricante ou ambos, route"/search_ivm", que representa os dados introduzidos no form e ao clicar procurar executa a query presente em app.cgi, route"/ivm" com os parâmetros passados e demonstra então uma tabela com as IVM com as características encontradas.*



5. Route"/ivm" - app.cgi

f. Sub-Categorias

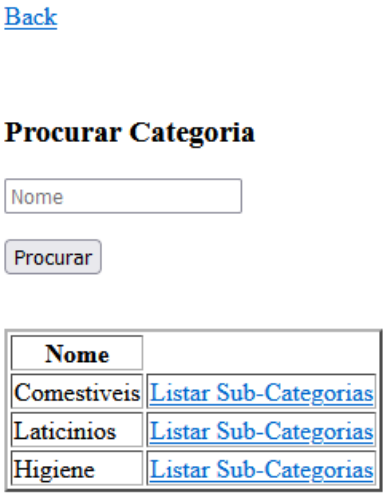
- *A página de Sub-Categorias, em app.cgi que renderiza super_cat.html, permite-nos consultar todas as Super Categorias presentes no sistema bem como apresentar todas as suas Sub-Categorias a todos os níveis de profundidade.Optámos por fazer isto da seguinte forma:*
    - *Listar Sub-Categorias aparece na tabela ao lado do Nome a Listar de forma a ser mais fácil de executar a ação;*
    - *Procurar uma IVM pode ser feito escrevendo o Nome, restringindo a tabela às Sub-Categorias com o Nome especificado.*

i. Listar Sub-Categorias

- *Ao entrar na opção "Listar Sub-Categorias" é renderizado o ficheiro show_sub_cats.html que nos apresenta uma tabela com o Nome de todas as Sub-Categorias associadas à Super Categoria correspondente na tabela.*

ii. Procurar

- *Ao clicar "Procurar" é renderizado o ficheiro search.html, através dos parâmetros passados em app.cgi, route"/search_supercategoria", que representa os dados introduzidos no form e ao clicar procurar executa a query presente em app.cgi, route"/list_super_categoria" com os parâmetros passados e demonstra então uma tabela com a Super Categoria procurada.*



6. Route"/super_categoria"- app.cgi

g. Sql Injection

- *De forma a tornar a nossa aplicação segura a ataques por Sql Injection, usámos verificações de input em diversas zonas do nosso programa. Para fazer estas verificações criámos a função isInvalidInput() em app.cgi que recebe uma string e retorna verdadeiro se esta for inválida. Neste caso remete o utilizador para uma página de erro, erro.html.*

```
1 def isInvalidInput(s):
2     return '--' in s or ';' in s or "'" in s or s == ""
```

7. Função "isInvalidInput" – app.cgi

```
1 nome = request.form["nome"]
2 if (isInvalidInput(nome)):
3     return render_template("erro.html",
4         spot='Categoria', campo='Nome', literal=nome)
```

8. Exemplo - route"updt_cat_add"- app.cgi

7. Índices

a. ***SELECT DISTINCT R.nome***
   FROM retalhista R, responsavel_por P
   WHERE R.tin = P.tin and P. nome_cat = 'Frutos'

- *CREATE INDEX index_responsavel_por ON responsavel_por USING BTREE(nome_cat);*
- *CREATE INDEX index_retalhista ON retalhista HASH(nome);*

    - *De modo a melhorar os tempos faria sentido criar dois indexes.*
    - *O primeiro na tabela responsavel_por sobre o atributo nome_cat, pois temos uma procura sequencial (Seq Scan), o que em tabelas de dimensão elevada seria uma operação demorada, neste caso a procura em árvore é mais rápida.*
    - *O segundo na tabela retalhista sobre o atributo nome, pois temos uma ordenação dos nomes na tabela retalhista no início de execução da query o que não seria necessário com hash.*

b. ***SELECT T.nome, count(T.ean)***
   FROM produto P, tem_categoria T
   WHERE p.cat = T.nome and P.desc like 'A%'
   GROUP BY T.nome

- *CREATE INDEX index_tem_categoria ON tem_categoria USING HASH(nome);*
- *CREATE INDEX index_produto ON produto USING BTREE(descr);*

    - *De modo a melhorar os tempos faria sentido criar dois indexes.*
    - *O primeiro na tabela tem_categoria sobre o atributo nome, pois isto tornaria a operação de join substancialmente mais célere.*
    - *O segundo na tabela produto sobre o atributo descr, pois temos uma procura sequencial (Seq Scan), o que em tabelas de dimensão elevada seria uma operação demorada, neste caso a procura em árvore é mais rápida.*