Projeto de Base de Dados

Base de Dados - L20

Grupo 115

Nome	Número	Esforço
Diogo Melita	99202	35h (33%)
João Rocha	99256	35h (33%)
Tiago Silva	99335	36h (34%)

Professores: João Aparício e Leonardo Alexandre

4º Período, 2º Semestre, Ano Letivo 2021/2022
Licenciatura em Engenharia Informática e de Computadores – Alameda
23 de junho de 2022

1 | Carregamento da Base de Dados

Para a resolução desta parte do projeto, desenvolvemos vários ficheiros com vista a criar a base de dados, seguindo os dados do enunciado:

- No ficheiro **schema.sql** temos toda a criação das tabelas;
- No ficheiro **ICs.sql** encontramos todas as restrições de integridade pedidas e algumas adicionais que achámos relevante para a resolução do projeto;
- No ficheiro **populate.sql** temos as instruções que introduzem os dados na base de dados.

Desta forma, para carregar a base de dados apenas temos de executar os seguintes comandos por ordem:

```
\i schema.sql
\i ICs.sql
\i populate.sql
```

2 | Restrições de Integridade

2.1 | Restrições de Integridade do Enunciado

Em relação às restrições de integridade pedidas, é relevante fazer o seguinte esclarecimento:

Para desenvolver a primeira RI considerámos apenas que uma categoria não podia estar contida nela própria diretamente, ou seja, não podia existir uma relação **has_other** onde os dois argumentos fossem iguais (como sugerido na RI-RE5 do modelo relacional no Anexo A do enunciado).
Para garantir isto optámos por um CHECK. Foi a única das IC's em que usámos um CHECK por ser a única que não requeria nenhuma query, e dependia apenas dos novos valores.

2.2 | Triggers Adicionais

Adicionámos ainda 3 triggers adicionais que tornam a utilização do site mais fácil.
O primeiro e o segundo adicionam uma categoria à tabela das categorias (se não existir) quando é acrescentada, respetivamente, uma categoria simples ou super.
A terceira adiciona também uma categoria mas à tabela de categorias simples caso tentemos adicionar um produto a uma categoria que ainda não existe.

3 | Queries SQL

Não há nenhum esclarecimento especial a fazer em relação a esta parte do projeto.

4 | Vista SQL

Na construção da vista usamos ISODOW em vez de DOW para termos o dia da semana entre Segunda (1) e Domingo (7) em vez de entre Domingo (0) e Sábado (6).

5 | Aplicação Web

Para o desenvolvimento da aplicação web recorremos a um script de Python CGI e ficheiros template HTML. Para além disso, para realizar a comunicação entre a aplicação e a base de dados decidimos, para as queries mais complexas, armazená-las em procedures que se encontram no ficheiro **"procedures.sql"**, sendo necessário correr o comando **\i procedures.sql** antes de se utilizar a app. Para além disso, utilizando estes procedures, que são atómicos, e a library **"psycopg2"**, através dos métodos *connect* e *commit*, garantimos a segurança contra SQL Injections. A nossa aplicação web quatro opções no menu principal: "Ver categorias simples", "Ver super-categorias", "Ver retalhistas" e "Ver eventos de reposição". Esta aplicação web está disponível em Aplicação Web - Grupo 115[1]

[1] https://web2.ist.utl.pt/ist199335/app.cgi/main_menu

5.1 | Ver categorias simples

Clicando na opção **"Ver categorias simples"**, é apresentada a tabela que lista todas as categorias simples, com a opção de remoção de cada elemento. Ao remover um elemento, é apresentado uma página que indica se a operação foi bem sucedida.
Para além de eliminar elementos, também existe a possibilidade de adicionar um elemento, através do botão **"Adicionar nova categoria"**, levando-nos para a página destinada ao efeito. Caso a categoria seja introduzida com sucesso, é apresentada ao utilizador uma mensagem a indicá-lo. Caso esta categoria já exista na base de dados, a operação não é efetuada e é apresentada uma página de erro.

5.2 | Ver super-categorias

Clicando no botão **Ver super-categorias** o utilizador é levado para a respetiva página. Esta página tem as mesmas carateristícas da página das categorias simples, com exceção da coluna na tabela que permite consultar as subcategorias de cada super-categoria, para todos os níveis de profundidade. Aqui, é nos possível adicionar e remover subcategorias à super-categoria. Ao adicionar uma subcategoria, ela será subcategoria direta da super-categoria. Caso a subcategoria não exista como categoria na base de dados, esta será adicionada como categoria simples. Ao remover uma subcategoria, caso esta não seja subcategoria direta, mas sim subcategoria indireta (subcategoria de uma subcategoria direta), irá ser apagada a relação entre a subcategoria indireta e a sua super-categoria.

5.3 | Ver retalhistas

Clicando no botão **Ver retalhistas**, somos levados para uma página onde são listados todos os retalhistas, podendo-se adicionar e remover retalhistas, assim como consultar as suas responsabilidades de reposição. Caso se tente adicionar dados (TIN e Nome) de um retalhista que já exista, irá ser apresentada ao utilizador uma página de erro. Caso contrário, irá ser apresentada uma página a indicar o sucesso da operação.
Carregando em **Responsabilidades**, somos redirecionados para uma página onde encontramos uma tabela com todas as responsabilidades do retalhista escolhido onde podemos adicionar e remover responsabilidades. Ao adicionar uma responsabilidade, caso um dos atributos (categora, número de série da IVM ou fabricante da IVM) não exista na base de dados, será apresentada uma mensagem de erro.

5.4 | Ver eventos de reposição

Por fim, clicando no último botão, **Ver eventos de reposição**, somos redirecionados para uma página que lista as diferentes IVMs, por **Nº de Série** e **Fabricante**, podendo-se consultar através do botão Consultar todos eventos de reposição de cada uma destas. Estes eventos de reposição irão estar ordenados por categorias.

6 | Consultas OLAP

Quanto às consultas OLAP, considerámos tanto na primeira como na segunda que nos pediam o agrupamento por cada característica individualmente (por concelho, por categoria, etc) e por todas em conjunto. Desta forma, o uso de CUBE e ROLLUP não era possível, pelo que foi necessário especificar os conjuntos pretendidos com GROUPING SETS.

7 | Indices

7.1 | Primeira Query

```
CREATE INDEX resp_cat_idx ON responsible_for(category_name) INCLUDE (tin);
CREATE INDEX retailer_idx ON retailer(tin) INCLUDE (name);
```

Na primeira query optámos por:

1. Um índice Btree na tabela **responsible_for**, com chave de procura **category_name**, que inclui o atributo **tin**.

Isto permite-nos obter o atributo **tin** de todas as linhas em P cuja categoria seja **Frutos** apenas acedendo ao índice e, consequentemente, sem ter de aceder tanto ao disco. Assim, aceleramos consideravelmente a nossa query, sem aumentar demasiado o overhead de espaço ocupado na memória primária.
Note-se que, apesar de termos uma condição de igualdade no WHERE, optamos por um índice Btree, pois, em Postgres, índices Hash não permitem a operação INCLUDE. Assim, o uso de índice Hash (apesar de ser mais eficiente a encontrar as entradas que satisfazem a condição WHERE) tornaria mais lenta a obtenção dos **tins** necessários.

2. Um índice Btree na tabela retailer, com chave de procura **tin**, que inclui o atributo **name**.
Esta opção é feita largamente pela mesma razão que a do índice anterior. Este índice permite-nos filtrar os retailers pelos **tins** obtidos no índice acima e obter os **names** respetivos, poupando acessoas a disco, acelerando consideravelmente a query.
É de notar que, neste caso, índices Btree sobre os atributos **tin** e **name** da tabela retailer já existem, por serem, respetivamente, chave principal e unique. Contudo, sem criar este novo índice, não conseguimos uma consulta "Index-Only" o que, por aceder menos a disco, é consideravelmente mais rápido.

7.2 | Segunda Query

```
CREATE INDEX prod_desc_idx ON product(descr) INCLUDE (cat);
CREATE INDEX has_cat_name_idx ON has_category(name) INCLUDE (ean);
```

Na segunda query optámos por:

1. Um índice Btree na tabela **product**, com chave de procura **descr**, que inclui o atributo **cat**.
Isto permite-nos fazer a comparação P.desc LIKE 'A%' (que pode ser entendida como P.desc >= 'A' AND P.desc < 'B') uma vez que Btree permite comparações de ordem. Ao incluirmos o atributo **cat** no índice, conseguimos obtê-lo sem acessos extra ao disco, tal como na primeira query.

2. Um índice Btree na tabela **has_category**, com chave de procura **name**, que inclui o atributo **ean**.
Este índice é útil visto que fazemos tanto um WHERE como um GROUP BY (e o SELECT) sobre esse atributo. A inclusão do atributo **ean**, mais uma vez, permite obter os **eans** (necessários para a projeção) com uma consulta "Index-Only", o que acelera bastante o tempo da query.