

Projeto de BD - Parte 3

Grupo 002 — Turno L13 — LEIC-A

Prof. Flávio Martins



Gonçalo Bárias (103124) - 33.33% - 50h

Raquel Braunschweig (102624) - 33.33% - 50h

Vasco Paisana (102533) - 33.33% - 50h

Desenvolvimento da Aplicação

Explicação da Aplicação

No desenvolvimento da nossa aplicação *web* procurámos para além do que nos foi solicitado criar uma **imagem que fosse apelativa**, desenvolver **novas funcionalidades** e ter como preocupação um *runtime* rápido, dando assim importância ao uso de **lógica de paginação** para as listas que apresentamos.

Na **página inicial**, podemos encontrar 2 opções: **customer view** e **manager view**. Clicando na opção **customer view** é possível fazer um novo registo ou, caso o cliente já esteja registado, o login. Nesta opção o cliente poderá fazer encomendas, ver as encomendas que estão pagas e as que estão por regularizar. Poderá ainda verificar os detalhes de cada encomenda e efetuar pagamentos.

Em **manager view** temos três campos: **customers**, **suppliers** e **products**. No **customers** podemos observar uma lista com todos os clientes. No **suppliers** é-nos dada a informação sobre os fornecedores e a possibilidade de registar novos. No **products** temos a informação sobre os produtos, podendo criar novos e alterar dados dos existentes. Finalmente podemos remover tanto clientes, como fornecedores e produtos.

Explicação do código

Na pasta **app** apresentamos um ficheiro **docker-compose.yml** que juntamente com o **Dockerfile** permitem correr a aplicação no endereço *web* **http://127.0.0.1:5001**. Deste modo, basta na *root* do diretório executar o comando **sudo docker-compose up -d** que vai executar não só a aplicação no endereço mencionado anteriormente, como também uma base de dados que é possível popular executando o ficheiro **run.sh** no terminal. Para parar os contentadores de **Docker** basta executar o comando **sudo docker-compose down**. Achámos que isto ajudaria com o processo de executar múltiplas vezes a aplicação.

Apresentamos na pasta **app/templates** todas as páginas html que em conjunto constroem o *frontend* do nosso projeto.

Cada tipo de dado (**customer**, **supplier**, **order** e **product**) tem as suas respetivas páginas html de registo e de listagem. No caso dos produtos, que pode ser acedido utilizando quer o **customer view**, quer a **manager view** procuramos diferenciar o tipo de utilizador, de modo a conseguir alterar o conteúdo que cada um vê. No caso do **customer view**, a página de listagem de produtos tem a funcionalidade de fazer encomendas, enquanto que o **manager view** permite adicionar, remover e editar produtos.

Pelo código, assim como na pasta **js**, temos alguns scripts que utilizamos de modo a conseguirmos implementar não só pop ups, como verificações de inputs de modo a mitigar o **error handling**.

De modo a simplificar a aplicação achámos também importante estabelecer uma série de **triggers** no ficheiro **ICs.sql** que removem as dependências das relações que são eliminadas pelo utilizador no *frontend*.

Índices

Foi pedido, ainda, que se indicasse (justificando) os índices que faria sentido criar, por forma a *agilizar* a execução de cada uma das *queries* apresentadas de seguida.

Primeira *query*:

```
1 SELECT order_no
2 FROM orders
3     JOIN contains USING (order_no)
4     JOIN product  USING (SKU)
5 WHERE price > 50 AND
6     EXTRACT(YEAR FROM date) = 2023;
```

Para a primeira *query*, optou-se por criar dois índices: um no atributo **price** da relação **product**, e outro no ano do atributo **date** da relação **orders**.

- Não foi criado nenhum índice para os atributos **order_no** e **SKU** de **contains**, pois o PostgreSQL já cria um índice **BTree** para (**order_no**, **SKU**) dado que se trata de uma chave primária. O **planner** é capaz de usar este índice e assim não se justifica criar dois índices separados para estes atributos apenas devido às cláusulas **JOIN**.

- Optámos pela criação de um índice **BTree** no atributo **price** de **product**, pois a comparação pretendida engloba um intervalo de preços e assim um índice **Hash** não seria particularmente inteligente, já que não se pretende um único preço em concreto.
- Por fim, criámos um índice para o ano do atributo **date** de **orders**, pois estando a fazer uma comparação de igualdade no ano das datas das encomendas, faz todo o sentido usar um índice **Hash** já que a comparação em $O(1)$ é ideal.

O trecho de código correspondente à indexação pretendida encontra-se abaixo:

```
1 DROP INDEX IF EXISTS product_price_index;
2 DROP INDEX IF EXISTS order_date_index;
3
4 CREATE INDEX product_price_index
5   ON product USING BTREE(price);
6 CREATE INDEX order_date_index
7   ON orders USING HASH(EXTRACT(YEAR FROM date));
```

Segunda *query*:

```
1 SELECT order_no, SUM(qty * price)
2 FROM contains
3   JOIN product USING (SKU)
4 WHERE name LIKE 'A%'
5 GROUP BY order_no;
```

Para a segunda *query*, optou-se por criar um índice no atributo **name** da relação **product**.

- Aqui a cláusula **GROUP BY** beneficia de um índice **BTree** em **order_no** de **contains**, pois o **GROUP BY** procura agrupar os dados sobre o atributo **order_no** e os índices **BTree** já vêm, por natureza, ordenados. Porém, tal como no ponto anterior, o **planner** é capaz de usar o índice criado na chave primária de **contains** para agilizar a computação do **GROUP BY**, sendo esse já um índice **BTree**.
- Assim, apenas é necessário criar um índice **BTree** para o atributo **name** de **product**, pois a comparação pretendida engloba todo um intervalo de nomes de produtos que começam pela letra A, logo um índice **Hash** não ajudaria.

O trecho de código correspondente à indexação pretendida encontra-se abaixo:

```
1 DROP INDEX IF EXISTS product_name_index;
2
3 CREATE INDEX product_name_index
4   ON product USING BTREE(name);
```