Projeto de BD - Parte 3

Grupo 47			
Nome	Nº	Esforço (Horas)	Contribuição (%)
Frederico Silva	99222	12	33.33%
Guilherme Leitão	99951	12	33.33%
Sebastião Carvalho	99326	12	33.33%

Turno Prático L19, Prof. Daniela Machado

Considerações:

Restringimo-nos de pôr toda a informação diretamente relacionada com o código da resolução deste projeto, pois pensámos que seria desnecessário repetir este duas vezes, uma vez que já se encontra disponível sob a forma de diversos ficheiros em anexo.

- SQL:

1. Decidimos considerar um inteiro o atributo loc da tabela planograma.
2. Não usámos “not null” em primary keys pois a criação destas já vem com esta restrição, bem como em atributos de aparente pouca relevância. No entanto, decidimos utilizar “not null” em todos os atributos que consideramos ser importantes no contexto do projeto.
3. Decidimos usar o tipo de dados numeric para keys com um número fixo de algarismos de forma a que o arredondamento de casas decimais seja mais preciso.
4. O carregamento de dados foi feito manualmente, visando a obtenção de uma boa resposta às interrogações propostas.

- Análise de Dados:

Como sugerido, na criação da vista “vendas”, foi utilizada a função “extract” para obter os valores desejados a partir da “timestamp”. Em particular, para o atributo dia_semana, usou-se para o “field value” o valor “dow” que representa o dia da semana de 0 a 6 (acabando no Sábado) ao invés do “isodow” que o representa de 1 a 7 (acabando no Domingo).

- Web:

No desenvolvimento da aplicação web foram nos dados 4 requisitos:

1) Inserir e remover categorias e sub-categorias; 2) Inserir e remover um retalhista, com todos os seus produtos, garantindo que esta operação seja atómica; 3) Listar todos os eventos de reposição de uma IVM, apresentando o número de unidades repostas por categoria de produto; 4) Listar todas as sub-categorias de uma super-categoria, a todos os níveis de profundidade.

Para isso, desenvolvemos uma aplicação web com 4 menus principais:
um menu inicial que contém todas as IVMS, com a possibilidade de listar os eventos de reposição de cada uma. Tem um menu para os retalhistas, com as funções de adicionar e remover retalhistas. Temos ainda um menu com categorias, que permite adicionar e remover categorias e adicionar sub-categorias e este tem uma ligação a um menu com todas as ligações entre categoria e sub-categoria.

Foram tomadas as seguintes decisões no desenvolvimento da aplicação web:

1. Quando ocorre a adição de uma categoria, ela é sempre considerada uma categoria simples.
2. Quando ocorre a remoção de uma sub-categoria, a super categoria nunca passa a ser categoria simples, continuando sempre na tabela super-categoria, embora possa não ter sub-categorias. Tomámos esta decisão supondo que super-categorias podem ter 0 sub-categorias, desde que tenham tido pelo menos uma vez uma sub-categoria.
3. Quando removemos uma sub-categoria, apenas é removida a relação entre super e sub-categoria e não a categoria em si.
4. Na listagem de sub-categorias a todos os níveis de profundidade, decidimos não usar funções recursivas de SQL, optando por usar uma breadth first search (BFS) para inserir todas as categorias numa lista, e fazendo um query final para obter todas as categorias. Esta decisão foi tomada para não usar keywords não dadas em aula.
5. Ao adicionar uma sub-categoria, esta tem que já existir, não podendo ser criada na altura.
6. Optámos por inserir a maior parte das hiperligações diretamente nas tabelas para minimizar os erros e garantir facilidade ao utilizador.
7. Inserimos sempre um menu de confirmação em qualquer evento de remoção, para melhorar a user experience e minimizar a ocorrência de erros de “missclick”.
8. Não foi usada a instrução “START TRANSACTION”, pois o software usado para a ligação à base de dados, o psycopg2, já cria uma transação sempre que começa uma ligação, e é utilizado o método “commit” no fim para enviar as alterações para a base de dados, garantindo sempre a atomicidade das operações.
9. Não foi usado nenhum css no desenvolvimento da aplicação, tendo-nos focado mais na funcionalidade da mesma do que na estética.

- Índices:

1) Sugerimos a implementação de um índice do tipo hash para os atributos nome_cat e tin da tabela responsavel_por, pois são interrogações com base em igualdades. No entanto, como apenas um deles é necessário, apenas deverá ser criado aquele que apresentar maior seletividade, visando agilizar ao máximo a query. Não é necessário um índice para o atributo tin da tabela retalhista pois este é uma primary key, pelo que já possui um índice. Da mesma forma, visto que o postgresql cria automaticamente um índice para colunas unique, este não é necessário implementar para o atributo nome da tabela retalhista.

```
CREATE INDEX nome_cat_idx ON responsavel_por USING HASH(nome_cat);

/* OR */

CREATE INDEX tin_idx ON responsavel_por USING HASH(tin);
```

2) (1) Sugerimos a implementação de um índice do tipo B+ tree para o atributo desc da tabela produto pois é uma interrogação com base num intervalo. (2) No entanto, também existe a possibilidade de criar um índice do tipo hash e B+ tree para os atributos cat e nome das tabelas produto e tem_categoria, respetivamente, pois estas tratam de interrogações com igualdades e, no caso de nome, um group by. Contudo, como no caso anterior, também aqui é necessária apenas uma das duas combinações apresentadas anteriormente, pelo que a escolha deve ser feita em prol da menor abundância de resultados (maior seletividade). Em adição, visto se tratar de uma função de agregação numa coluna, é também aconselhável um índice do tipo B+ tree para o atributo ean da tabela tem_categoria.

```
3) CREATE INDEX desc_idx ON produto(descr);
4) CREATE INDEX ean_idx ON tem_categoria(ean);
5)
6) /* OR */
7)
8) CREATE INDEX cat_idx ON produto USING HASH(cat);
9) CREATE INDEX nome_ean_idx ON tem_categoria(nome, ean);
```