Projeto BD - Parte 3

Grupo 105

Turno BD2L15 - Profª Daniela Machado

Nome	Número	Contribuição	Esforço total (h)
Francisco Abrunhosa	95580	33%	8
Inês Magessi	95593	33%	8
João Silveira	95597	33%	8

1. Base de dados

Notas acerca da interpretação:

- Relativamente a produto e suas categorias, interpretamos que a coluna “cat” na relação “Produto” designa a categoria mais específica a que o produto pertence. Isto é, se essa categoria do produto for uma subcategoria de outra/s subcategoria/s, então o produto pertence diretamente à categoria mais abaixo na hierarquia das categorias. Posto isto, e dado que a associação “has_category” representa uma relação many to many, interpretamos também que em “has_category” há uma entrada para cada par (produto, categoria) para todas as categorias na hierarquia de categorias do produto. Foi também por este motivo que decidimos, contrariamente ao representado no enunciado, colocar “ean” e “nome” como primary keys de “has_category”, dado que consideramos ser uma relação many-to-many.
  Exemplo:
  - produto1 pertence à categoria simples “Refrigerantes com Gás”
  - A categoria “Refrigerantes com Gás” é uma subcategoria de “Refrigerantes” que por sua vez é uma subcategoria de “Bebidas”
  - Na relação “Produto” há uma entrada cujo ean é 1, descrição “produto1” e cat “Refrigerantes com Gás”
  - Na relação “has_category” há uma entrada para cada par (produto, categoria), ou seja, uma entrada (1, “Refrigerantes com Gás”), outra (1, “Refrigerantes”) e por fim, uma (1, “Bebidas”)

Plantamos a base de dados tendo em conta esta assunção, pelo que, para a restrição de integridade (RI-5), não temos de fazer uma recursão na relação “has_other” de forma a determinar a hierarquia de categorias a que o produto pertence. Podemos usar diretamente a informação guardada em “has_category”.

- A relação “responsible_for” no enunciado indica apenas “num_serie” e “fabricante” como primary keys. Isto significa que, para cada IVM pode haver apenas um retalhista responsável por repor uma categoria. Ou seja, nenhuma das outras categorias dentro da IVM teria um repositor. Dado que não nos faz sentido que assim o seja, decidimos considerar “nome_cat” e “tin” como primary keys também. Assim, passa a ser possível que para cada categoria numa IVM haja apenas um retalhista por ela responsável, mas esse mesmo retalhista pode ser responsável por outras categorias dentro da mesma IVM.
  Posto isto, se cumprissemos com o explícito no enunciado, teríamos de criar tantas IVM quanto categorias simples e colocar um retalhista responsável por uma categoria simples em cada uma dessas IVM para que a query SQL nº 2 tenha um resultado não nulo. Pelo contrário, fazendo segundo a nossa interpretação, temos apenas de colocar um retalhista responsável por cada uma das categorias simples em qualquer IVM, podendo mesmo ser responsável por mais do que uma na mesma máquina.

5. Desenvolvimento da Aplicação

- A aplicação está dividida em três partes, uma para cada relação:
  - Eventos de Reposição
    - **/replenishments/** - Lista os eventos de reposição, dada um query de procura
  - Categorias
    - **/categories/** - Página inicial das categorias que serve simultaneamente para listar as categorias
    - **/categories/create/** - Página dedicada à criação de novas categorias e à adição de subcategorias. A escolha do form a preencher e submeter é feita com JavaScript no lado do cliente.
    - **/categories/delete/** - Página dedicada à remoção de categorias. A escolha do form a preencher e submeter é feita com JavaScript no lado do cliente.
  - Retalhistas
    - **/retailers/** - Página inicial dos retalhistas
    - **/retailers/create/** - Página dedicada à criação de um novo retalhista.
    - **/retailers/delete/** - Página dedicada à remoção de um retalhista.

Foram, também, feitas algumas escolhas para manter a coerência da base de dados de acordo com o enunciado.

- Durante a criação de uma super categoria o utilizador é forçado a adicionar uma primeira subcategoria, pois não faz sentido existirem super categorias vazias.
- Durante a criação de um retalhista o utilizador é forçado a adicionar uma relação "responsible_for", uma vez que não faz sentido haver retalhistas sem responsabilidades, de acordo com o enunciado.

O link para aplicação é http://web.tecnico.ulisboa.pt/ist195597/app.cgi. No entanto, experienciámos algumas dificuldades ao tentar correr a aplicação porque os serviços de web da DSI estavam em baixo. De qualquer modo, é possível correr a aplicação localmente usando o Flask. As instruções para correr a aplicação desta forma encontram-se disponíveis no README.md na pasta web do zip entregue.

6. Consultas OLAP

Notas acerca da interpretação:

- Na segunda consulta interpretamos o pedido como sendo: para um dado distrito, o número total de artigos vendidos por parâmetro individualmente. Ou seja, por distrito e concelho, distrito e categoria, distrito e dia da semana; nunca fazendo um cruzamento da informação sobre concelho, categoria e dia da semana numa mesma entrada.

- Na primeira consulta decidimos fazer uma filtragem inicial da informação para ficarmos apenas com entradas do período especificado e só depois agrupar pelos parâmetros pedidos. Mais uma vez, interpretamos que não há cruzamento de informação entre os parâmetros. Ou seja, há duas colunas de agrupamento (dia da semana e concelho) e a informação é agregada tendo em conta cada uma individualmente.
  Para filtrar a data, tentamos inicialmente concatenar a informação apresentada nas colunas do ano, mês e dia de forma a criar uma só data numa cadeia de caracteres e poder compará-la com as datas de input. No entanto, como temos entradas na tabela cujo mês em formato timestamp tem apenas um dígito e outras entradas em que o mês tem dois dígitos, a comparação de strings não funciona corretamente. Posto isto, optamos por fazer uma query mais longa e menos elegante mas que de facto funciona em todas as situações.

7. Índices

- Primeira consulta:

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

CREATE INDEX nome_index ON retalhista(nome);
```

Esta query seria otimizada com índices em R.tin, P.tin e P.nome_cat. No entanto R.tin é uma chave primária, e P.tin e P.nome_cat são chaves estrangeiras, pelo que não há necessidade de criar índices sobre estes atributos visto que já estão implícitos. Apesar disso, a cláusula SELECT DISTINCT remove duplicados, ou seja, há um sorting, pelo que deve haver um índice em R.nome.

- Segunda consulta:

```
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 desc_index ON produto USING HASH (desc);
```

Esta query seria optimizada com índices em P.cat e T.nome, no entanto P.cat e T.nome são chaves estrangeiras, e como tal não precisam que se crie índices dado que já são criados por defeito. Ainda assim, é benéfico criar um índice do tipo hash em P.desc para otimizar a cláusula WHERE P.desc like 'A%'.