Instituto Superior Técnico Ano Letivo 2021/2022

Projeto BD – Parte 3

Grupo 79 - Turno B2L07

Professor Rodrigo Sousa

Aluno	Número de aluno	Esforço (horas)
Manuel Pereira	98580	10 Horas (33%)
Rita Costa	95908	10 Horas (33%)
Tiago Peralta	99332	10 Horas (33%)

Instituto Superior Técnico Ano Letivo 2021/2022

Arquitetura da Aplicação Web

Link: http://web2.ist.utl.pt/ist198580/app.cgi/

Inserir Categorias:

- É pedido o nome da categoria a inserir. no campo “Nova Categoria”;
- Insere a categoria como categoria simples;

Remover Categorias:

- É pedido o nome da categoria a remover, no campo “Categoria a remover”;
- É verificado se é uma sub-categoria;
- Caso se trate de uma sub-categoria, é feita uma verificação para ver quantas sub-categorias é que a super-categoria da categoria a remover tem;
- Caso seja a única sub-categoria dessa super-categoria, movemos essa super-categoria para as categorias simples;
- No fim, remove-se todas as referências a categoria, que se pretende remover, na base de dados;

Inserir Sub-Categorias:

- É pedido o nome da nova sub-categoria e o nome da super-categoria;
- Começamos por verificar se a super-categoria já existe na base de dados;
- Se a super-categoria já existir como categoria simples, movemos para a tabela das super-categorias;
- Caso a super-categoria não exista na base de dados, é inserida como super-categoria;
- Depois é feita uma verificação se a sub-categoria já existe na base de dados;
- Caso não exista, é adicionada como um categoria simples;

- No fim, é adicionada uma relação entre ambas a super-categoria e a sub-categoria na tabela tem_outra;

Remover Sub-Categorias:

- É pedido o nome da sub-categoria a remover;
- Verificamos se a super-categoria associada a essa sub-categoria tem mais sub-categorias;
- Caso a sub-categoria a remover seja a única sub-categoria da sua super-categoria, então movemos a super-categoria para as categorias simples;
- No fim, apenas eliminamos a relação entre a sub-categoria e a super-categoria na tabela tem_outra;

Inserir Retalhistas:

- É pedido o tin e o nome do retalhista;
- Insere o tin e o nome na tabela retalhista;

Acrescentar Responsabilidade:

- É pedido o nome da categoria, o tin do retalhista, o número de série e o fabricante de uma IVM livre;
- É adicionada uma responsabilidade do retalhista, associado ao tin, sobre a categoria, na IVM associada ao número de série e fabricante, na tabela responsavel_por;

Remover Retalhistas:

- É pedido o tin do retalhista;
- Remove-se todas as referências ao tin, que se pretende remover, na base de dados;

Listar Eventos de Reposição de uma IVM:

- É pedido o número de série e o fabricante da IVM que se pretende verificar os eventos de reposição;

- A primeira tabela apresentada lista todos os eventos de reposição associados a essa IVM;

- A segunda tabela mostra a quantidade de unidades respostas, nessa ivm, agrupada por categoria;

Listar Sub-Categorias de uma Super-Categoria:

- É pedido o nome da super-categoria;

- É apresentada uma lista de todas as sub-categorias dessa super-categoria, a todos os níveis de profundidade;

Relações entre Arquivos

populate.sql e web/app.cgi:

- Foram criados diferentes stored functions e procedures para permitir as funcionalidades da aplicação;

- Para a funcionalidade de inserir categorias foi criado o procedure inserir_categoria;

- Para a funcionalidade de remover categorias foi criado o procedure remover_categoria;

- Para a funcionalidade de inserir sub-categorias foi criado o procedure inserir_subcategoria;

- Para a funcionalidade de remover sub-categorias foi criado o procedure remover_subcategorias;

- Para a funcionalidade de listar as sub-categorias de uma super-categoria, foi criado a stored function listar_subcategorias;

populate.sql e Cls.sql:

- O código do procedimento para a terceira restrição de integridade, utiliza a stored function listar_subcategorias, para obter todas as subcategorias da categoria de uma prateleira;

Consultas OLAP

Consulta 1:

- O período considerado foi entre as datas 28/10/2017 e 21/02/2022;

Consulta 2:

- O distrito considerado foi o “DISTRITO_15”;

Índices

Query 1:

SQL:

CREATE INDEX cat_idx **ON** responsavel_por **USING HASH**(nome_cat);
CREATE INDEX rtin_idx **ON** retalhista **USING HASH**(tin);
CREATE INDEX ptin_idx **ON** responsavel_por **USING HASH**(tin);

Explicação:

- Começamos por partir do pressuposto que tanto a tabela retalhista como a tabela responsavel_por têm maioritariamente operações de leitura;

- Como existe uma seleção entre duas tabelas e depois temos R.tin=P.tin, estamos perante um join;

- De forma a realizar o join da forma mais eficiente possível, criamos dois índices hash para o atributo tin, nas tabelas retalhista e responsavel_por (o que também respeita a boa prática de criar índices para chaves primárias), de forma a que o join seja feito através de um hash join;

- Como na tabela na retalhista o nome já tem automaticamente um índice criado, por ser UNIQUE, não faz sentido criar nenhum;

- Resta o atributo nome_cat, onde já faz sentido criar um índice Hash devido a igualdade P.nome_cat='Frutos', pois permite não ter de percorrer toda a tabela responsavel_por, sendo apenas necessário pesquisar num “bucket”;

Alternativas consideradas:

- Também consideramos a hipótese de não criar um índice hash para tin tanto na tabela retalhista como na tabela responsavel_por;

- A alternativa seria aproveitar o índice que é criado automaticamente pelo SGBD, por tin ser chave primária na tabela retalhista, percorrendo todos os registos de tin na tabela responsavel_por e fazendo a verificação da igualdade R.tin=P.tin, através do índice criado;

- Decidimos não seguir esta alternativa pelos seguintes motivos:

    - A eficiência temporal compensa o acréscimo de espaço em disco;

    - Os custo de manutenção de ter mais um índice não vai ser muito elevado, pois temos como pressuposto que a maioria das operações sobre as tabelas são de leitura;

Query 2:

SQL:

```
CREATE INDEX desc_idx ON produto USING BTREE(desc);
CREATE INDEX cat_idx ON produto USING HASH(tin);
CREATE INDEX nome_idx ON tem_categoria USING HASH(nome);
```

Explicação:

- Começamos por partir do pressuposto que tanto a tabela produto como a tabela tem_categoria têm maioritariamente operações de leitura;

- Como existe uma seleção entre duas tabelas e depois temos p.cat=T.nome, estamos perante um join;

- De forma a realizar o join da forma mais eficiente possível, criamos dois índices hash para o atributo cat e para o atributo nome, nas tabelas produto (o que também respeita a boa prática de criar índices para chaves primárias), e tem_categoria respectivamente, de forma a que o join seja feito através de um hash join;

- Para o atributo desc na tabela produto faz sentido criar um índice B+ tree, devido a P.desc LIKE 'A%', pois as folhas do índice vão estar ordenadas e como os registos que pretendemos começam com ‘A’, vamos chegar rapidamente aos registos, sem ter de percorrer toda a tabela;

Alternativas consideradas:

- Também consideramos a hipótese de não criar um índice hash para cat na tabela produto como para nome na tabela tem_categoria;

- A alternativa seria aproveitar o índice que é criado automaticamente pelo SGBD, por nome ser chave primária na tabela tem_categoria, percorrendo todos os registos de cat na tabela produto e fazendo a verificação da igualdade p.cat=T.nome, através do índice criado;

- Decidimos não seguir esta alternativa pelos seguintes motivos:

    - A eficiência temporal compensa o acréscimo de espaço em disco;

    - Os custo de manutenção de ter mais um índice não vai ser muito elevado, pois temos como pressuposto que a maioria das operações sobre as tabelas são de leitura;