Dados Projeto de BD – Parte 3

Grupo 136 – Turno BD2L14

Professor Flávio Martins

Aluno	Contribuição	Esforço Total
André Azevedo (92424)	34%	13 horas
Maria Gomes (97856)	33%	13 horas
Sara Marques (93342)	33%	13 horas

1. Base de Dados

Esquema (populate.sql)

```sql
DROP TABLE IF EXISTS categoria cascade;
DROP TABLE IF EXISTS categoria_simples cascade;
DROP TABLE IF EXISTS super_categoria cascade;
DROP TABLE IF EXISTS tem_outra cascade;
DROP TABLE IF EXISTS produto cascade;
DROP TABLE IF EXISTS tem_categoria cascade;
DROP TABLE IF EXISTS IVM cascade;
DROP TABLE IF EXISTS ponto_de_retalho cascade;
DROP TABLE IF EXISTS instalada_em cascade;
DROP TABLE IF EXISTS prateleira cascade;
DROP TABLE IF EXISTS planograma cascade;
DROP TABLE IF EXISTS retalhista cascade;
DROP TABLE IF EXISTS responsavel_por cascade;
DROP TABLE IF EXISTS evento_reposicao cascade;

-------------------------------------
-- Table Creation
-------------------------------------
-- Named constraints are global to the database.
-- Therefore the following use the following naming rules:
--   1. pk_table for names of primary key constraints
--   2. fk_table_another for names of foreign key constraints

CREATE TABLE categoria (
    nome VARCHAR(255) NOT NULL UNIQUE,
    CONSTRAINT pk_categoria PRIMARY KEY(nome)
);

CREATE TABLE categoria_simples (
    nome VARCHAR(255) NOT NULL UNIQUE,
    CONSTRAINT pk_categoria_simples PRIMARY KEY(nome),
    CONSTRAINT fk_categoria_simples_categoria FOREIGN KEY(nome) REFERENCES
categoria(nome)
);

CREATE TABLE super_categoria (
    nome VARCHAR(255) NOT NULL UNIQUE,
    CONSTRAINT pk_super_categoria PRIMARY KEY(nome),
    CONSTRAINT fk_super_categoria_categoria FOREIGN KEY(nome) REFERENCES categoria(nome)
);

CREATE TABLE tem_outra (
    super_categoria VARCHAR(255) NOT NULL,
    categoria VARCHAR(255) NOT NULL UNIQUE,
    CONSTRAINT pk_tem_outra PRIMARY KEY(categoria),
    CONSTRAINT fk_tem_outra_super_categoria FOREIGN KEY(super_categoria) REFERENCES
super_categoria(nome),
```

```
    CONSTRAINT fk_tem_outra_categoria FOREIGN KEY(categoria) REFERENCES categoria(nome)
);

CREATE TABLE produto (
    ean NUMERIC(13, 0) NOT NULL UNIQUE,
    descr VARCHAR(255) NOT NULL,
    cat VARCHAR(255) NOT NULL,
    CONSTRAINT pk_produto PRIMARY KEY(ean),
    CONSTRAINT fk_produto_categoria FOREIGN KEY(cat) REFERENCES categoria(nome)
);

CREATE TABLE tem_categoria (
    ean NUMERIC(13, 0) NOT NULL,
    nome VARCHAR(255) NOT NULL,
    CONSTRAINT pk_tem_categoria PRIMARY KEY(ean, nome),
    CONSTRAINT fk_tem_categoria_produto FOREIGN KEY(ean) REFERENCES produto(ean),
    CONSTRAINT fk_tem_categoria_categoria FOREIGN KEY(nome) REFERENCES categoria(nome)
);

CREATE TABLE IVM (
    num_serie VARCHAR(255) NOT NULL,
    fabricante VARCHAR(255) NOT NULL,
    CONSTRAINT pk_IVM PRIMARY KEY(num_serie, fabricante)
);

CREATE TABLE ponto_de_retalho (
    nome VARCHAR(255) NOT NULL UNIQUE,
    distrito VARCHAR(255) NOT NULL,
    concelho VARCHAR(255) NOT NULL,
    CONSTRAINT pk_ponto_de_retalho PRIMARY KEY(nome)
);

CREATE TABLE instalada_em (
    num_serie VARCHAR(255) NOT NULL,
    fabricante VARCHAR(255) NOT NULL,
    "local" VARCHAR(255) NOT NULL,
    CONSTRAINT pk_instalada_em PRIMARY KEY(num_serie, fabricante),
    CONSTRAINT fk_instalada_em_IVM FOREIGN KEY(num_serie, fabricante) REFERENCES
IVM(num_serie, fabricante),
    CONSTRAINT fk_instalada_em_ponto_de_retalho FOREIGN KEY("local") REFERENCES
ponto_de_retalho(nome)
);

CREATE TABLE prateleira (
    nro INTEGER NOT NULL,
    num_serie VARCHAR(255) NOT NULL,
    fabricante VARCHAR(255) NOT NULL,
    altura NUMERIC(4,2),
    nome VARCHAR(255) NOT NULL,
    CONSTRAINT pk_prateleira PRIMARY KEY(nro, num_serie, fabricante),
```

```
    CONSTRAINT fk_prateleira_IVM  FOREIGN KEY(num_serie, fabricante) REFERENCES
IVM(num_serie, fabricante),
    CONSTRAINT fk_prateleira_categoria FOREIGN KEY(nome) REFERENCES categoria(nome)
);

CREATE TABLE planograma (
    ean NUMERIC(13, 0) NOT NULL,
    nro INTEGER NOT NULL,
    num_serie VARCHAR(255) NOT NULL,
    fabricante VARCHAR(255) NOT NULL,
    faces INTEGER NOT NULL,
    unidades INTEGER NOT NULL,
    CONSTRAINT pk_planograma PRIMARY KEY(ean, nro, num_serie, fabricante),
    CONSTRAINT fk_planograma_produto FOREIGN KEY(ean) REFERENCES produto(ean),
    CONSTRAINT fk_planograma_prateleira FOREIGN KEY(nro, num_serie, fabricante)
REFERENCES prateleira(nro, num_serie, fabricante)
);

CREATE TABLE retalhista (
    tin NUMERIC(9,0) NOT NULL UNIQUE,
    "name" VARCHAR(255) NOT NULL UNIQUE,
    CONSTRAINT pk_retalhista PRIMARY KEY(tin)
);

CREATE TABLE responsavel_por (
    nome_cat VARCHAR(255) NOT NULL,
    tin NUMERIC(9,0) NOT NULL,
    num_serie VARCHAR(255) NOT NULL,
    fabricante VARCHAR(255) NOT NULL,
    CONSTRAINT pk_responsavel_por PRIMARY KEY(num_serie, fabricante),
    CONSTRAINT fk_responsavel_por_IVM FOREIGN KEY(num_serie, fabricante) REFERENCES
IVM(num_serie, fabricante),
    CONSTRAINT fk_responsavel_por_retalhista FOREIGN KEY(tin) REFERENCES retalhista(tin),
    CONSTRAINT fk_responsavel_por_categoria FOREIGN KEY(nome_cat) REFERENCES
categoria(nome)
);

CREATE TABLE evento_reposicao (
    ean NUMERIC(13,0) NOT NULL,
    nro INTEGER NOT NULL,
    num_serie VARCHAR(255) NOT NULL,
    fabricante VARCHAR(255) NOT NULL,
    instante TIMESTAMPTZ NOT NULL,
    unidades INTEGER NOT NULL,
    tin INTEGER NOT NULL,
    CONSTRAINT pk_evento_reposicao PRIMARY KEY(ean, nro, num_serie, fabricante,
instante),
    CONSTRAINT fk_evento_reposicao_planograma FOREIGN KEY(ean, nro, num_serie,
fabricante) REFERENCES planograma(ean, nro, num_serie, fabricante),
    CONSTRAINT fk_evento_reposicao_retalhista FOREIGN KEY(tin) REFERENCES retalhista(tin)
)
```

2. Restrições de Integridade

(ICs.sql)

```
DROP TRIGGER IF EXISTS chk_categoria_ciclo_trigger ON tem_outra;
DROP TRIGGER IF EXISTS chk_unidades_evento_reposicao_trigger ON evento_reposicao;
DROP TRIGGER IF EXISTS chk_produto_prateleira_categoria_trigger ON planograma;


--(RI-1) Uma Categoria nao pode estar contida em si propria
CREATE OR REPLACE FUNCTION chk_categoria_ciclo_proc() RETURNS TRIGGER AS
$$
BEGIN
    IF NEW.super_categoria = NEW.categoria THEN
        RAISE EXCEPTION 'Uma Categoria nao pode estar contida em si propria';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER chk_categoria_ciclo_trigger
BEFORE UPDATE OR INSERT ON tem_outra
FOR EACH ROW EXECUTE PROCEDURE chk_categoria_ciclo_proc();


--(RI-4) O numero de unidades repostas num Evento de Reposicao não pode exceder o numero
de
--unidades especificado no Planograma
CREATE OR REPLACE FUNCTION chk_unidades_evento_reposicao_proc() RETURNS TRIGGER AS
$$
DECLARE unidades_planograma INTEGER;
BEGIN
    SELECT planograma.unidades INTO unidades_planograma
    FROM planograma
    WHERE planograma.ean = NEW.ean AND planograma.nro = NEW.nro AND planograma.num_serie
= NEW.num_serie
          AND planograma.fabricante = NEW.fabricante;

    IF NEW.unidades > unidades_planograma THEN
        RAISE EXCEPTION 'O numero de unidades repostas num Evento de Reposicao nao pode
exceder o numero de unidades especificado no Planograma';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER chk_unidades_evento_reposicao_trigger
BEFORE UPDATE OR INSERT ON evento_reposicao
FOR EACH ROW EXECUTE PROCEDURE chk_unidades_evento_reposicao_proc();
```

```sql
--(RI-5) Um Produto so pode ser reposto numa Prateleira que apresente (pelo menos) uma
das
--Categorias desse produto
CREATE OR REPLACE FUNCTION chk_produto_prateleira_categoria_proc() RETURNS TRIGGER AS
$$
DECLARE count_categorias INTEGER;
BEGIN
    SELECT COUNT(*) INTO count_categorias
    FROM prateleira P
    WHERE P.nro = NEW.nro AND P.num_serie = NEW.num_serie AND P.fabricante =
NEW.fabricante AND P.nome IN (
        SELECT nome
        FROM tem_categoria T
        WHERE T.ean = NEW.ean
    );

    IF count_categorias = 0 THEN
        RAISE EXCEPTION 'Um Produto so pode ser reposto numa Prateleira que apresente
(pelo menos) uma das Categorias desse produto';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER chk_produto_prateleira_categoria_trigger
BEFORE UPDATE OR INSERT ON planograma
FOR EACH ROW EXECUTE PROCEDURE chk_produto_prateleira_categoria_proc();
```

3. SQL

(queries.sql)

```sql
-- Qual o nome do retalhista (ou retalhistas) responsaveis pela reposicao do maior numero
de categorias?

SELECT "name"
FROM retalhista
    NATURAL JOIN (
        SELECT tin, COUNT(*)
        FROM responsavel_por
        GROUP BY tin
        HAVING COUNT(*) >= ALL (
            SELECT COUNT(*)
            FROM responsavel_por
            GROUP BY tin
        )
    ) AS retalhista_mais_responsavel;
```

```sql
-- Qual o nome do ou dos retalhistas que sao responsaveis por todas as categorias
simples?

SELECT DISTINCT "name"
FROM retalhista
INNER JOIN responsavel_por ON retalhista.tin = responsavel_por.tin
INNER JOIN categoria_simples ON categoria_simples.nome = responsavel_por.nome_cat;


-- Quais os produtos (ean) que nunca foram repostos?

SELECT ean
FROM produto
WHERE ean NOT IN (
    SELECT ean
    FROM evento_reposicao);


-- Quais os produtos (ean) que foram repostos sempre pelo mesmo retalhista?

SELECT ean
FROM evento_reposicao
GROUP BY ean
HAVING COUNT(DISTINCT tin) = 1;
```

4. Vistas

(view.sql)

```sql
DROP VIEW IF EXISTS Vendas;

CREATE VIEW Vendas(ean, cat, ano, trimestre, mes, dia_mes, dia_semana, distrito,
concelho, unidades)
AS
SELECT evento_reposicao.ean,
    produto.cat,
    EXTRACT(YEAR FROM instante) AS ano,
    EXTRACT(QUARTER FROM instante) AS trimestre,
    EXTRACT(MONTH FROM instante) AS mes,
    EXTRACT(DAY FROM instante) AS dia_mes,
    EXTRACT(DOW FROM instante) AS dia_semana,
    ponto_de_retalho.distrito,
    ponto_de_retalho.concelho,
    evento_reposicao.unidades
FROM evento_reposicao
INNER JOIN produto ON evento_reposicao.ean = produto.ean
INNER JOIN instalada_em ON evento_reposicao.num_serie = instalada_em.num_serie AND
evento_reposicao.fabricante = instalada_em.fabricante
INNER JOIN ponto_de_retalho ON instalada_em.local = ponto_de_retalho.nome;
```

5. Desenvolvimento da Aplicação

A arquitetura da aplicação web começa com um menu(index.html), no qual existem quatro opções, nomeadamente “Categorias”, “Retalhistas”, “Eventos de Reposição” e “Sub-Categorias” que correspondem respetivamente às quatro funcionalidades que nos são pedidas no enunciado para o protótipo.

Nas categorias(categorias.html) é apresentado uma tabela com todas categorias que existem na base de dados e a sua respetiva super-categoria. Se a categoria não tiver uma super-categoria então é exibido uma string vazia que significa que o valor da célula é null.
Para além da listar as categorias é possível inserir e remover categorias. Caso o utilizador pretenda remover uma categoria existe uma “form” na linha de cada categoria que ao ser pressionada a faz apagar. O processo de apagar uma categoria ao nível da app.cgi é feito de forma análoga ao que o comando “ON DELETE CASCADE” faz automaticamente, em que eliminamos primeiro os registos das tabelas que têm uma “FOREIGN KEY CONSTRAINT” na tabela categoria e nas subsequentes tabelas que forem alvo de eliminação.
Para inserir uma categoria na tabela existe um link no topo que leva o utilizador a uma nova página(inserir_categoria.html) e tem uma “form” que é necessário preencher para poder submeter e inserir uma nova categoria na tabela. Por questão de correção, quando se escolhe uma super-categoria apenas são dadas como opções as categorias que estão efetivamente na tabela super_categoria e a string vazia para representar que a categoria não tem super-categoria.

A opção dos retalhistas(retalhistas.html) segue os mesmo padrão que as categoria. A tabela apresenta exclusivamente os atributos do retalhista. As ideias ao inserir e remover categorias aplicam-se igualmente aos retalhistas. No entanto para cada linha da tabela com um retalhista existe também um link(responsabilidades.html) que leva às responsabilidades do utilizador. A página de responsabilidades apresenta uma tabela com os atributos exclusivos de responsavel_por para um dado retalhista, ou seja, a página tem dados diferentes para cada retalhista selecionado. É usado parâmetros no url para determinar qual o retalhista selecionado e apresentar as suas responsabilidades. Na página das responsabilidades é permitido inserir e remover responsabilidades. O processo é outra vez semelhante ao das categorias, destaque apenas para quando estamos a preencher a “form” para inserir uma responsabilidade em “inserir_responsabilidades.html” são dadas como opções apenas IVMs que estão disponíveis, e cada opção tem os atributos num_serie e fabricante intrinsecamente associados. Isto porque num_serie e fabricante são “PRIMARY KEY” de responsavel_por e não seria correto estar a inserir uma IVM que não fosse única ou com um número de série de um fabricante que não fosse o seu.

Ao aceder os Eventos de Reposição são apresentadas primeiro todas as IVMs presentes na base de dados. Esta página(ivms.html) serve de passagem á verdadeira funcionalidade da opção em listar todos os todos os eventos de reposição de uma IVM, apresentando o número de unidades respostas por categoria de produto. É usado parâmetros no url para determinar qual a ivm a ser selecionada na página(eventos_reposicao.html) e apresentar os seus dados.

Para as Sub-Categorias a lógica é a mesma dos Eventos de Reposição, em que apresentamos primeiro todas as Super-Categorias. Na página(super_categorias.html) conseguimos aceder às respetivas sub-categorias de cada super-categoria a partir dos parâmetros no url e na página(sub_categorias.html) são listadas todas as suas sub-categorias, a todos os níveis de profundidade.
O processo de listar todas as sub-categorias de uma super-categoria é feito com base em obter as sub-categorias diretas da super-categoria e ciclicamente obter todas as outras sub-categorias de cada sub-categoria direta, até que não existam mais sub-categorias das sub-categorias já identificadas.

Em cada página é possível voltar á página anterior segundo a estrutura da aplicação, ao contrário da funcionalidade “Back button” do browser que acede à última página visitada.

Link: https://web2.ist.utl.pt/ist192424/app.cgi/

7. Índices

7.1-

Como a Primary Key do responsavel_por é composta com as colunas (num_serie, fabricante) ou seja, não inclui nem o tin nem o nome_cat, é necessário criar um índice para estas colunas de forma a tornar este query mais rápido. Visto tratarem se de operações de igualdade o melhor tipo de índice será do tipo hash (hash(tin) e hash(nome_cat)).

```
CREATE index idx_responsavel_por_tin ON responsavel_por USING hash(tin);
CREATE index idx_responsavel_por_nome_cat ON responsavel_por USING
hash(nome_cat);
```

7.2-

Como a Primary Key do produto é o ean, é necessário criar um índice para o atributo cat de forma a tornar o query mais rápido. Como se trata de uma operação de igualdade o melhor tipo de índice será do tipo hash (hash(cat))

Sendo que a Primary key da consulta é do tipo Btree e que esta é uma chave composta com as colunas (ean, nome), uma reordenação destas colunas com o nome como primeira coluna da chave seria suficiente para acelerar a execução deste query, btree(nome, ean).

Para além disso, tendo em conta que queremos as descrições dos produtos começadas pela letra ‘A’, usar uma Btree em que esta está organizada alfabeticamente seria uma forma de otimizar a query.

```
CREATE index idx_produto_cat ON produto USING hash(cat);
CREATE index idx_tem_categoria_nome ON tem_categoria USING hash(nome);
CREATE index idx_produto_desc ON produto(desc);
```