BASES DE DADOS

Miguel Eleutério – 99287 | Raquel Cardoso – 99314 | Tiago Ferreira – 99334

Prof.ª Daniela Falcão Machado

Alunos	Esforço	Horas
Miguel Eleutério	33.3%	25
Raquel Cardoso	33.3%	25
Tiago Ferreira	33.3%	25
Total:	100%	75

GRUPO 121 – BD2L19
24 de Junho de 2022

1. Base de Dados

```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;


CREATE TABLE categoria(
nome VARCHAR(40) NOT NULL,
CONSTRAINT pk_categoria PRIMARY KEY(nome)
);

CREATE TABLE categoria_simples(
nome VARCHAR(40) NOT NULL,
CONSTRAINT pk_categoria_simples PRIMARY KEY(nome),
CONSTRAINT fk_categoria_simples FOREIGN KEY(nome) REFERENCES categoria(nome)
);

CREATE TABLE super_categoria(
nome VARCHAR(40) NOT NULL,
CONSTRAINT pk_super_categoria PRIMARY KEY(nome),
CONSTRAINT fk_super_categoria FOREIGN KEY(nome) REFERENCES categoria(nome)
);

CREATE TABLE tem_outra(
super_categoria VARCHAR(40) NOT NULL,
categoria VARCHAR(40) NOT NULL,
CONSTRAINT pk_tem_outra PRIMARY KEY(categoria),
CONSTRAINT fk1_tem_outra FOREIGN KEY(super_categoria) REFERENCES super_categoria(nome),
CONSTRAINT fk2_tem_outra FOREIGN KEY(categoria) REFERENCES categoria(nome)
);
CREATE TABLE produto(
ean INT NOT NULL,
cat VARCHAR(40) NOT NULL,
descr VARCHAR(40) NOT NULL,
CONSTRAINT pk_produto PRIMARY KEY(ean),
CONSTRAINT fk_produto FOREIGN KEY(cat) REFERENCES categoria(nome)
);

CREATE TABLE tem_categoria(
ean INT NOT NULL,
nome VARCHAR(40) NOT NULL,
CONSTRAINT fk1_tem_categoria FOREIGN KEY(ean) REFERENCES produto(ean),
CONSTRAINT fk2_tem_categoria FOREIGN KEY(nome) REFERENCES categoria(nome)
);
```

```sql
CREATE TABLE IVM(
num_serie CHAR(20) NOT NULL  ,
fabricante VARCHAR(40) NOT NULL ,
CONSTRAINT pk_IVM PRIMARY KEY(num_serie,fabricante)
);

CREATE TABLE ponto_de_retalho(
nome VARCHAR(40) NOT NULL ,
distrito VARCHAR(40) NOT NULL,
concelho VARCHAR(40) NOT NULL,
CONSTRAINT pk_ponto_de_retalho PRIMARY KEY(nome)
);

CREATE TABLE instalada_em(
num_serie CHAR(20) NOT NULL ,
fabricante VARCHAR(40) NOT NULL ,
local VARCHAR(40) NOT NULL ,
CONSTRAINT pk_instalada_em PRIMARY KEY(num_serie,fabricante),
CONSTRAINT fk1_instalada_em FOREIGN KEY(num_serie,fabricante) REFERENCES IVM(num_serie,fabricante),
CONSTRAINT fk2_instalada_em FOREIGN KEY(local) REFERENCES ponto_de_retalho(nome)
);

CREATE TABLE prateleira(
nro INT NOT NULL ,
num_serie CHAR(20) NOT NULL ,
fabricante VARCHAR(40) NOT NULL ,
altura INT NOT NULL,
nome VARCHAR(40) NOT NULL,
CONSTRAINT pk_prateleira PRIMARY KEY(nro,num_serie,fabricante),
CONSTRAINT fk1_prateleira FOREIGN KEY(num_serie,fabricante) REFERENCES IVM(num_serie,fabricante),
CONSTRAINT fk2_prateleira FOREIGN KEY(nome) REFERENCES categoria(nome)
);

CREATE TABLE planograma(
ean INT NOT NULL,
nro INT NOT NULL ,
num_serie CHAR(20) NOT NULL ,
fabricante VARCHAR(40) NOT NULL ,
faces INT NOT NULL,
unidades INT NOT NULL,
loc VARCHAR(40) NOT NULL,
CONSTRAINT pk_planograma PRIMARY KEY(ean,nro,num_serie,fabricante),
CONSTRAINT fk1_planograma FOREIGN KEY(ean) REFERENCES produto(ean),
CONSTRAINT fk2_planograma FOREIGN KEY(num_serie,fabricante,nro) REFERENCES prateleira(num_serie,fabricante,nro)
);
CREATE TABLE retalhista(
tin INT NOT NULL,
nome VARCHAR(40) NOT NULL,
CONSTRAINT pk_retalhista PRIMARY KEY(tin)
);

CREATE TABLE responsavel_por(
nome_cat VARCHAR(40) NOT NULL,
tin INT NOT NULL,
num_serie CHAR(20) NOT NULL ,
fabricante VARCHAR(40) NOT NULL ,
CONSTRAINT pk_responsavel_por PRIMARY KEY(num_serie,fabricante),
CONSTRAINT fk1_responsavel_por FOREIGN KEY(num_serie,fabricante) REFERENCES IVM(num_serie,fabricante),
CONSTRAINT fk2_responsavel_por FOREIGN KEY(tin) REFERENCES retalhista(tin),
CONSTRAINT fk3_responsavel_por FOREIGN KEY(nome_cat) REFERENCES categoria(nome)
);

CREATE TABLE evento_reposicao(
ean INT NOT NULL,
nro INT NOT NULL ,
tin INT NOT NULL,
num_serie CHAR(20) NOT NULL,
fabricante VARCHAR(40) NOT NULL,
instante DATE NOT NULL,
unidades INT NOT NULL,
CONSTRAINT pk_evento_reposicao PRIMARY KEY(ean,nro,num_serie,fabricante,instante),
CONSTRAINT fk1_evento_reposicao FOREIGN KEY(ean,nro,num_serie,fabricante) REFERENCES planograma(ean,nro,num_serie,fabricante),
CONSTRAINT fk2_evento_reposicao FOREIGN KEY(tin) REFERENCES retalhista(tin)
);
```

2. Restrições de Integridade

```sql
CREATE OR REPLACE FUNCTION check_category()
RETURNS TRIGGER AS
$$
BEGIN
  IF NEW.categoria = NEW.super_categoria THEN
    RAISE EXCEPTION 'Uma categoria nao pode estar contida nela propria';
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

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


-----------------------------------------------
CREATE OR REPLACE FUNCTION check_unidades()
RETURNS TRIGGER AS
$$
BEGIN
  IF NOT EXISTS (SELECT FROM planograma WHERE (NEW.nro = planograma.nro AND NEW.num_serie = planograma.num_serie
    AND NEW.ean = planograma.ean AND NEW.unidades <= planograma.unidades)) THEN
    RAISE EXCEPTION 'O numero de unidades repostas num evento de reposicao nao pode exceder o numero de unidades especificadas
    pelo planograma';
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER tr_check_unidades
BEFORE UPDATE OR INSERT ON evento_reposicao
FOR EACH ROW EXECUTE PROCEDURE check_unidades();

----------------------------------------
CREATE OR REPLACE FUNCTION check_cat_prateleira()
RETURNS TRIGGER AS
$$
BEGIN
  IF NOT EXISTS (SELECT ean FROM produto WHERE (produto.cat IN(SELECT nome FROM
    prateleira WHERE NEW.nro = prateleira.nro) AND NEW.ean = produto.ean)) THEN
    RAISE EXCEPTION 'Um produto so pode ser reposto numa prateleira que apresente uma das categorias desse produto';
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

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

3. SQL

```sql
-- EX 1
SELECT nome FROM retalhista WHERE retalhista.tin IN (
    SELECT tin, COUNT(DISTINCT nome_cat) FROM responsavel_por GROUP BY tin
    HAVING COUNT(DISTINCT nome_cat) >= ALL(
        SELECT COUNT(DISTINCT nome_cat) FROM responsavel_por GROUP BY tin
    )
);

-- EX 2
SELECT nome FROM retalhista NATURAL JOIN responsavel_por WHERE responsavel_por.nome_cat IN
(SELECT nome FROM categoria_simples )
GROUP BY tin HAVING COUNT(*) = (SELECT COUNT(*) FROM categoria_simples);

-- EX 3
SELECT ean FROM produto WHERE produto.ean NOT IN(SELECT ean FROM evento_reposicao);

-- EX 4
SELECT ean FROM produto WHERE produto.ean IN(SELECT ean FROM(
    SELECT a.* FROM evento_reposicao a INNER JOIN(SELECT ean, fabricante, COUNT(*)
totalCOUNT FROM evento_reposicao GROUP BY ean,fabricante HAVING COUNT(*) > 1)
b ON a.fabricante = b.fabricante AND a.ean = b.ean) AS ean);
```

4. Vistas

```
CREATE VIEW vendas(ean, cat, ano, trimestre, mes, dia_mes, dia_semana, distrito, concelho, unidades) AS
SELECT ean, cat, EXTRACT(YEAR FROM instante) AS ano, EXTRACT(MONTH FROM instante) as mes,
EXTRACT(QUARTER FROM instante) AS trimestre, EXTRACT(DAY FROM instante) AS dia_mes,
EXTRACT(DOW FROM instante) AS dia_semana, distrito, concelho, unidades FROM
produto NATURAL JOIN ponto_de_retalho NATURAL JOIN evento_reposicao;
```

5. Desenvolvimento da Aplicação

Link para versão de trabalho: **https://web2.tecnico.ulisboa.pt/ist199287/web-app.cgi/**

Para o desenvolvimento do protótipo de aplicação requisitada, necessitámos de criar um script Python CGI (web-app.cgi) e alguns ficheiros HTML (dentro da pasta templates). A função *index*() na app, que dá *return* à interpretação visual do ficheiro *index.html,* corresponde à raiz da nossa aplicação e é onde encontramos o necessário para cumprir as quatro alíneas:

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 **inserir e remover categorias** clicamos em “Mostrar categorias” que nos irá levar para a função *list_categories*() que irá devolver o ficheiro *categories.html* onde nos são apresentadas todas as Categorias da base de dados. No final da tabela podemos selecionar entre Adicionar ou Remover uma categoria. Ao escolher adicionar vamos para a função *add_category*() que nos irá devolver o ficheiro *add_category.html* para escolher um nome para a nova categoria. Ao clicar em enviar iremos para a função *add_category_to_database()* onde caso seja um nome válido irá executar a *query* para adicionar a categoria à *database*. Uma categoria adicionada por este método será ***automaticamente*** definida como categoria simples. De seguida iremos retornar para a função *index*() que nos trará de volta para a página inicial. Para remover uma categoria será o mesmo processo para funções e ficheiros de nome semelhante.

Para **inserir e remover retalhistas** clicamos em “Mostrar categorias” que irá por sua vez nos levar para a função *list_retailers*() que irá devolver a representação visual do ficheiro *retailers.html*, apresentando uma lista de todos os Retalhistas. Para adicionar e remover retalhistas iremos seguir um processo semelhante ao acima descrito. Para garantir que a operação remover um retalhista seja atómica removemos da base de dados todas as relações de responsabilidade do mesmo e ***também*** os seus eventos de reposição.

Para **listar todos os eventos de reposição de uma IVM** clicamos em “Mostrar eventos de reposição” que nos levará para a função *list_reposition_events*() que nos mostrará o ficheiro *reposition_events*.html onde irá ver todos os eventos de reposição existentes. De seguida, na função *select_IVM*() é nos devolvido o template do ficheiro *select_ivm*.html onde é pedido ao utilizador para selecionar em dois menus de *dropdown* o número de série e fabricante da IVM da qual deseja ver os eventos de reposição. Uma vez que uma cada IVM tem sempre um par composto por número de série mais fabricante, ***não eliminamos*** os repetidos de qualquer um destes menus. Após, vamos para a função *specific_IVM*() que irá apresentar o

ficheiro *specific_ivm.html,* onde numa tabela mostrará todos os eventos de reposição da IVM e noutra mostrará o número de unidades repostas por categoria de produto, facilitando a compreensão do significado da coluna “Total”.

Para **listar todas as sub-categorias de uma super-categoria**, iremos clicar em “*Mostrar hierarquia*” que nos levará para a função *list_hierarchy*() que irá devolver visualmente o ficheiro *hierarchy*.html que mostra todas as relações “*tem_outra*” na base de dados. Para ver uma super-categoria específica iremos clicar em “Selecionar Super Categoria”, que, como no exercício anterior será pedido para selecionar o nome da Super Categoria com recurso à função *select_super_category()* e ao ficheiro *select_super_category.html*. Ao selecionar uma Super Categoria na base de dados iremos para a função *list_specific_hierarchy*() que irá devolver a representação visual do ficheiro *specific_hierarchy.html* que mostrará a tabela a todos os níveis de profundidade das sub-categorias da super-categoria pretendida.

7. Índices

1. CREATE INDEX idx_nomecat_tin ON responsavel_por USING hash(nome_cat, tin) WHERE nome_cat = “frutos”;
   CREATE INDEX idx_nome ON retalhista (nome);

Uma vez que o tin é uma primary key na tabela “retalhista”, por omissão tem um índice *Btree* associado. Desta forma, podemos optar por manter o índice Btree, ou criar um índice *hash*, uma vez que existe uma igualdade na query apresentada.

2. CREATE INDEX idx_cat ON produto USING hash(cat);
   CREATE INDEX idx_desc ON produto(desc) USING btree
   CREATE INDEX idx_nome_ean ON tem_categoria (nome, ean);

Criámos o índice *hash* porque a query exibida tem uma igualdade e criámos um índice *Btree* pois temos “P.desc like ‘A%’” e dessa forma sabemos que irá sempre começar por A, podendo restringir a nossa pesquisa e tornando o índice útil. Como a query utiliza tanto o nome como o ean, podemos agilizar este processo indexando as duas como um par.