



Projeto de Bases de Dados, Parte 3

83533	Mihail Brinza	.%	h
83557	Ricardo Brancas	.%	16h
83883	David Nunes	.%	h

Grupo 37, turno BD2251795L10
Professor Miguel Amaral

1 Criação da Base de Dados

```
-- Drop tables if they already exist. Will delete all data

DROP TABLE IF EXISTS reposicao;
DROP TABLE IF EXISTS evento_reposicao;
DROP TABLE IF EXISTS planograma;
DROP TABLE IF EXISTS prateleira;
DROP TABLE IF EXISTS corredor;
DROP TABLE IF EXISTS fornece_sec;
DROP TABLE IF EXISTS produto;
DROP TABLE IF EXISTS fornecedor;
DROP TABLE IF EXISTS constituida;
DROP TABLE IF EXISTS super_categoria;
DROP TABLE IF EXISTS categoria_simples;
DROP TABLE IF EXISTS categoria;

DROP TYPE IF EXISTS SIDE;
DROP TYPE IF EXISTS HEIGHT;

-- Create tables

CREATE TYPE SIDE AS ENUM ('esquerda', 'direita');
CREATE TYPE HEIGHT AS ENUM ('chao', 'medio', 'superior');

CREATE TABLE categoria (
    nome VARCHAR(80) NOT NULL,
    PRIMARY KEY (nome)
);

CREATE TABLE categoria_simples (
    nome VARCHAR(80) NOT NULL,
    PRIMARY KEY (nome),
    FOREIGN KEY (nome) REFERENCES categoria (nome)
);

CREATE TABLE super_categoria (
    nome VARCHAR(80) NOT NULL,
    PRIMARY KEY (nome),
    FOREIGN KEY (nome) REFERENCES categoria (nome)
);

CREATE TABLE constituida (
    super_categoria VARCHAR(80) NOT NULL,
    categoria VARCHAR(80) NOT NULL,
    PRIMARY KEY (super_categoria, categoria),
    FOREIGN KEY (super_categoria) REFERENCES super_categoria (nome),
    FOREIGN KEY (categoria) REFERENCES categoria (nome),
    CHECK (super_categoria <> categoria)
);

CREATE TABLE fornecedor (
    nif NUMERIC(9) NOT NULL,
    nome VARCHAR(80) NOT NULL,
    PRIMARY KEY (nif),
    CHECK (nif > 0)
);

CREATE TABLE produto (
    ean NUMERIC(13) NOT NULL,
    design VARCHAR(200) NOT NULL,
    categoria VARCHAR(80) NOT NULL,
    forn_primario NUMERIC(9) NOT NULL,
    data TIMESTAMP NOT NULL,
    PRIMARY KEY (ean),
    FOREIGN KEY (categoria) REFERENCES categoria (nome),
    FOREIGN KEY (forn_primario) REFERENCES fornecedor (nif),
    CHECK (ean > 0)
);
```

```

CREATE TABLE fornece_sec (
    nif NUMERIC(9) NOT NULL,
    ean NUMERIC(13) NOT NULL,
    PRIMARY KEY (nif, ean),
    FOREIGN KEY (nif) REFERENCES fornecedor (nif),
    FOREIGN KEY (ean) REFERENCES produto (ean)
);

CREATE TABLE corredor (
    nro SMALLINT NOT NULL,
    largura SMALLINT NOT NULL,
    PRIMARY KEY (nro),
    CHECK (nro > 0),
    CHECK (largura > 0)
);

CREATE TABLE prateleira (
    nro SMALLINT NOT NULL,
    lado SIDE NOT NULL,
    altura HEIGHT NOT NULL,
    PRIMARY KEY (nro, lado, altura),
    FOREIGN KEY (nro) REFERENCES corredor (nro)
);

CREATE TABLE planograma (
    ean NUMERIC(13) NOT NULL,
    nro SMALLINT NOT NULL,
    lado SIDE NOT NULL,
    altura HEIGHT NOT NULL,
    face SMALLINT NOT NULL,
    unidades SMALLINT NOT NULL,
    loc SMALLINT NOT NULL,
    PRIMARY KEY (ean, nro, lado, altura),
    FOREIGN KEY (ean) REFERENCES produto (ean),
    FOREIGN KEY (nro, lado, altura) REFERENCES prateleira (nro, lado, altura),
    CHECK (face > 0),
    CHECK (unidades > 0),
    CHECK (loc > 0)
);

CREATE TABLE evento_reposicao (
    operador VARCHAR(80) NOT NULL,
    instante TIMESTAMP NOT NULL,
    PRIMARY KEY (instante),
    UNIQUE (operador),
    UNIQUE (operador, instante),
    CHECK (instante <= CURRENT_TIMESTAMP)
);

CREATE TABLE reposicao (
    ean NUMERIC(13) NOT NULL,
    nro SMALLINT NOT NULL,
    lado SIDE NOT NULL,
    altura HEIGHT NOT NULL,
    operador VARCHAR(80) NOT NULL,
    instante TIMESTAMP NOT NULL,
    unidades SMALLINT NOT NULL,
    PRIMARY KEY (ean, nro, lado, altura),
    UNIQUE (operador),
    UNIQUE (instante),
    FOREIGN KEY (ean, nro, lado, altura)
        REFERENCES planograma (ean, nro, lado, altura),
    FOREIGN KEY (operador, instante)
        REFERENCES evento_reposicao (operador, instante),
    CHECK (unidades > 0)
);

CREATE OR REPLACE FUNCTION remove_cat(name VARCHAR(80))

```

```

        RETURNS VOID AS $$
DECLARE temp_name VARCHAR(80);
BEGIN
    SET CONSTRAINTS ALL DEFERRED;

    DELETE FROM constituida WHERE categoria = name;
    DELETE FROM categoria_simples WHERE nome = name;
    DELETE FROM categoria WHERE nome = name;

    FOR temp_name IN SELECT *
                        FROM super_categoria AS outter
                        WHERE NOT EXISTS(SELECT *
                                         FROM constituida
                                         WHERE super_categoria = outter.nome)
    LOOP
        INSERT INTO categoria_simples VALUES (temp_name);
        DELETE FROM super_categoria WHERE nome = temp_name;
    END LOOP;

    SET CONSTRAINTS ALL IMMEDIATE;
END
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION sub_cat_insert(super_name VARCHAR(80), sub_name VARCHAR(80))
RETURNS VOID AS $$
BEGIN
    SET CONSTRAINTS ALL DEFERRED;

    IF NOT EXISTS(SELECT * FROM categoria WHERE nome = super_name) THEN
        RAISE EXCEPTION 'A_super_categoria_escolhida_nao_existe.';
    END IF;

    IF EXISTS(SELECT nome FROM categoria_simples WHERE nome = super_name) THEN
        DELETE FROM categoria_simples WHERE nome = super_name;
        INSERT INTO super_categoria VALUES (super_name);
    END IF;

    INSERT INTO categoria VALUES (sub_name);
    INSERT INTO categoria_simples VALUES (sub_name);
    INSERT INTO constituida VALUES (super_name, sub_name);

    SET CONSTRAINTS ALL IMMEDIATE;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION all_subcategories(nome VARCHAR(80))
RETURNS SETOF VARCHAR(80) AS $$
DECLARE temp_name VARCHAR(80);
BEGIN
    DROP TABLE IF EXISTS result;
    DROP TABLE IF EXISTS temp_names;

    CREATE TEMP TABLE result (nome VARCHAR(80));

    CREATE TEMP TABLE temp_names AS
        SELECT categoria
        FROM constituida
        WHERE super_categoria = nome;

    WHILE EXISTS(SELECT * FROM temp_names)
    LOOP
        FOR temp_name IN SELECT * FROM temp_names
        LOOP
            INSERT INTO temp_names SELECT categoria FROM constituida
                                WHERE super_categoria = temp_name;
        END LOOP;
    END LOOP;
END

```

```

        DELETE FROM temp_names WHERE categoria = temp_name;

        INSERT INTO result VALUES (temp_name);
    END LOOP;
END LOOP;

RETURN QUERY SELECT * FROM result;
END;
$$ LANGUAGE plpgsql;

```

2 SQL

```

-- a
SELECT nome
FROM (
    SELECT forn_primario AS nif, categoria
    FROM produto
    UNION ALL
    SELECT nif, categoria
    FROM fornece_sec
    NATURAL JOIN produto
) AS c
NATURAL JOIN fornecedor
GROUP BY nif, nome
HAVING COUNT(DISTINCT categoria) >= ALL (
    SELECT MAX(count)
    FROM (
        SELECT nif, COUNT(DISTINCT categoria) AS count
        FROM (SELECT forn_primario AS nif, categoria
            FROM produto
            UNION ALL
            SELECT nif, categoria
            FROM fornece_sec
            NATURAL JOIN produto) AS a
        GROUP BY nif) AS b
    );

-- b
SELECT DISTINCT nif, nome
FROM produto AS out
JOIN fornecedor ON forn_primario = nif
WHERE NOT EXISTS(
    SELECT nome
    FROM categoria_simples
    EXCEPT
    SELECT categoria AS nome
    FROM produto
    NATURAL JOIN fornece_sec
    WHERE nif = out.forn_primario
    OR forn_primario = out.forn_primario
);

-- c
SELECT ean
FROM produto
EXCEPT
SELECT ean
FROM reposicao;

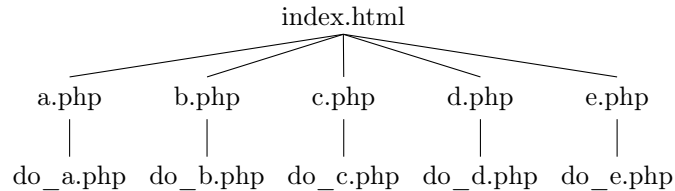
-- d
SELECT ean
FROM fornece_sec
GROUP BY ean
HAVING COUNT(nif) > 10;

-- e
SELECT ean
FROM reposicao
GROUP BY ean
HAVING COUNT(DISTINCT operador) = 1;

```

3 Desenvolvimento da Aplicação

A nossa aplicação em PHP segue o modelo típico cliente-servidor, em que a parte cliente se liga e faz pedidos ao SGBD via a interface PDO (PHP Data Objects). A página inicial, `index.html`, contém hiperligações para 5 outras páginas, cada uma correspondente a uma das alíneas pedidas; tal como demonstrado no esquema seguinte:



Para fazer os pedidos às respetivas páginas de execução (`do_*.php`) utilizámos o verbo GET no caso das alíneas **c** e **e**, por se tratar de um pedido sem “efeitos secundários” e POST nos restantes, por realizarem modificações à base de dados.

De um modo geral, em cada uma das páginas de input, existem formulários que o utilizador preenche e que, ao serem submetidos, enviam um pedido para a respetiva página de ação. Nessa página verificamos sumariamente o input do utilizador e executamos os *statements* correspondentes, utilizando transações quando se trata de uma ação *multi-statement* e fazendo *rollback* quando necessário.

No fim, em caso de sucesso, é mostrado ao utilizador qual o output, caso exista, e as *queries* que foram executadas. Se acontecer um erro, o mesmo é mostrado.

Descrevemos em seguida os detalhes de funcionamento para as alíneas relevantes:

a) Inserir e remover categorias e sub-categorias;

Na página de input (`a.php`) existem 3 opções disponíveis para o utilizador: inserir uma categoria (simples), inserir uma subcategoria ou remover uma categoria.

Consoante o formulário em que o utilizador faça *submit*, diferentes parâmetros vão ser passados no POST o que vai corresponder a ações diferentes realizadas no `do_a.php`.

b) Inserir e remover um novo produto e os seus respetivos fornecedores (primários ou secundários) garantindo que esta operação seja atômica;

São, novamente, dadas duas opções ao utilizador: inserir ou remover o produto. Tal como na alínea anterior, a form submetida define qual a ação a ser realizada.