# PROJETO BD – PARTE 3

Grupo 99, turno BD2L16, professor Daniel Mateus Gonçalves

| Número | Nome | Esforço total | Percentagem relativa de contribuição |
|---|---|---|---|
| 95531 | Ana Rita Duarte | 24 / 3 = 8 horas | 33.(3)% |
| 95572 | Filipa Cotrim | 24 / 3 = 8 horas | 33.(3)% |
| 95618 | Leonor Barreiros | 24 / 3 = 8 horas | 33.(3)% |

Nota: todo o projeto foi feito em conjunto

## Base de Dados

O arquivo correspondente a este componente da avaliação é o ficheiro populate.sql.

```
drop table evento_reposicao;
drop table responsavel_por;
drop table retalhista;
drop table planograma;
drop table prateleira;
drop table instalada_em;
drop table ponto_de_retalho;
drop table IVM;
drop table tem_categoria;
drop table produto;
drop table tem_outra;
drop table super_categoria;
drop table categoria_simples;
drop table categoria;

----------------------------------------
-- 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(100) not null unique,
  constraint pk_categoria primary key(nome)
);

create table categoria_simples (
  nome varchar(100) not null unique,
  constraint pk_categoria_simples primary key(nome),
  constraint fk_categoria_simples foreign key(nome) references
categoria(nome)
);

create table super_categoria (
  nome varchar(100) not null unique,
  constraint pk_super_categoria primary key(nome),
  constraint fk_super_categoria foreign key(nome) references categoria(nome)
);

create table tem_outra (
  super_categoria varchar(100) not null,
  categoria varchar(100) not null unique,
  constraint pk_tem_outra primary key(categoria),
  constraint fk_tem_outra foreign key(categoria) references categoria(nome)
);

create table produto (
  ean char(13) not null unique,
  cat varchar(100) not null,
  descr varchar(100) not null,
  constraint pk_produto primary key(ean),
  constraint fk_produto foreign key(cat) references categoria(nome)
);
```

```
create table tem_categoria (
  ean char(13) not null,
  nome varchar(100) not null,
  constraint fk_tem_categoria primary key(ean, nome),
  constraint fk_tem_categoria_nome foreign key(nome) references
categoria(nome),
  constraint fk_tem_categoria_ean foreign key(ean) references produto(ean)
);

create table IVM (
  num_serie int not null,
  fabricante varchar(100),
  constraint pk_IVM primary key(num_serie, fabricante)
);

create table ponto_de_retalho (
  nome varchar(100) not null unique,
  distrito varchar(100) not null,
  concelho varchar(100) not null,
  constraint pk_ponto_de_retalho primary key(nome)
);

create table instalada_em (
  num_serie int not null,
  fabricante varchar(100) not null,
  localization varchar(100) 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(localization)
references ponto_de_retalho(nome)
);

create table prateleira (
  nro int not null,
  num_serie int not null,
  fabricante varchar(100) not null,
  altura int not null,
  nome varchar(100) not null,
  constraint pk_prateleira primary key(nro, num_serie, fabricante),
  constraint pk_prateleira_categoria foreign key(nome) references
categoria(nome),
  constraint pk_prateleira_ivm foreign key(num_serie,fabricante) references
IVM(num_serie,fabricante)
);

create table planograma (
  ean char(13) not null,
  nro int not null,
  num_serie int not null,
  fabricante varchar(100) not null,
  faces int not null,
  unidades int not null,
  loc varchar(100) 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 varchar(100) not null unique,
  name varchar(100) not null unique,
  constraint pk_retalhista primary key(tin)
);

create table responsavel_por (
  nome_cat varchar(100) not null,
  tin varchar(100) not null,
  num_serie int not null,
  fabricante varchar(100) 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 char(13) not null,
  nro int not null,
  num_serie int not null,
  fabricante varchar(100) not null,
  instante timestamp not null,
  unidades int not null,
  tin varchar(100) 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)
);
```

## Restrições de integridade

O arquivo correspondente a este componente da avaliação é o ficheiro ICs.sql

```
DROP TRIGGER IF EXISTS not_contain_trigger ON tem_outra;
DROP TRIGGER IF EXISTS not_exceed_trigger ON evento_reposicao;
DROP TRIGGER IF EXISTS has_category_trigger ON evento_reposicao;
```

**(RI-1) – Uma Categoria não pode estar contida em si própria:**

```
CREATE OR REPLACE FUNCTION not_contain() RETURNS TRIGGER AS
$$
BEGIN
    IF NEW.super_categoria = NEW.categoria THEN
        RAISE EXCEPTION 'Category % cannot contain itself', NEW.categoria;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER not_contain_trigger
BEFORE UPDATE OR INSERT ON tem_outra
```

```
FOR EACH ROW EXECUTE PROCEDURE not_contain();
```

**(RI-4) – O número de unidades repostas num Evento de Reposição não pode exceder o número de unidades especificado no Planograma:**

```
CREATE OR REPLACE FUNCTION not_exceed() RETURNS TRIGGER AS
$$
DECLARE max_unidades INTEGER;
BEGIN
    SELECT unidades
    INTO max_unidades
    FROM planograma
    WHERE ean = NEW.ean AND nro = NEW.nro AND num_serie = NEW.num_serie AND
fabricante = NEW.fabricante;
    IF NEW.unidades > max_unidades THEN
        RAISE EXCEPTION 'Number of replaced units exceeds the maximum allowed
by planogram: %', max_unidades;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER not_exceed_trigger
BEFORE UPDATE OR INSERT ON evento_reposicao
FOR EACH ROW EXECUTE PROCEDURE not_exceed();
```

**(RI-5) – Um Produto só pode ser reposto numa Prateleira que apresente (pelo menos) uma das Categorias desse produto:**

```
CREATE OR REPLACE FUNCTION has_category() RETURNS TRIGGER AS
$$
DECLARE
    categoria VARCHAR(100);
    categoria_prat VARCHAR(100);
    cursor_categorias CURSOR FOR SELECT nome FROM tem_categoria WHERE ean =
NEW.ean;
BEGIN
    SELECT nome
    INTO categoria_prat
    FROM prateleira
    WHERE nro = NEW.nro AND num_serie = NEW.num_serie AND fabricante =
NEW.fabricante;
    -- iterar sobre as categorias do produto
    OPEN cursor_categorias;
    LOOP
        FETCH cursor_categorias INTO categoria;
        EXIT WHEN NOT FOUND;
        -- se a categoria for igual a da prateleira, OK :)
        IF categoria = categoria_prat THEN
            RETURN NEW;
        END IF;
    END LOOP;
    CLOSE cursor_categorias;
    -- chegando aqui, nao encontrou :(
    RAISE EXCEPTION 'The product must be replaced in a shelf which presents
one of its categories';
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER has_category_trigger
BEFORE UPDATE OR INSERT ON evento_reposicao
FOR EACH ROW EXECUTE PROCEDURE has_category();
```

## SQL

O arquivo correspondente a este componente da avaliação é o ficheiro queries.sql.

**Qual o nome do retalhista (ou retalhistas) responsáveis pela reposição do maior número de categorias?**

```
SELECT name
FROM retalhista NATURAL JOIN responsavel_por
GROUP BY name
HAVING COUNT(nome_cat) >= ALL
       (SELECT COUNT(nome_cat)
       FROM retalhista NATURAL JOIN responsavel_por
             GROUP BY name);
```

**Qual o nome do ou dos retalhistas que são responsáveis por todas as categorias simples?**

```
-- Projetar o nome dos retalhistas
SELECT R.name
FROM (
    -- Obter todos os retalhistas que correspondem
    SELECT *
    FROM retalhista
    -- A chave do retalhista tem de ser igual a chave de todos...
    WHERE tin = ALL (
        SELECT Q.tin
        FROM (
            -- ... os responsaveis por uma categoria simples
            SELECT *
            FROM responsavel_por
            WHERE nome_cat IN (
                SELECT nome FROM categoria_simples
            )
        ) AS Q
    )
) AS R;
```

**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;
```

## Vistas

O arquivo correspondente a este componente da avaliação é o ficheiro view.sql.

```
DROP VIEW IF EXISTS Vendas;
CREATE VIEW Vendas(ean, cat, ano, trimestre, mes, dia_mes, dia_semana,
distrito, concelho, unidades)
AS(
       SELECT produto.ean AS ean,
              categoria.nome AS cat,
              EXTRACT(YEAR FROM evento_reposicao.instante) AS ano,
              EXTRACT(QUARTER FROM evento_reposicao.instante) AS trimestre,
              EXTRACT(MONTH FROM evento_reposicao.instante) AS mes,
```

```
            EXTRACT(DAY FROM evento_reposicao.instante) AS dia_mes,
            EXTRACT(DOW FROM evento_reposicao.instante) AS dia_semana,
            ponto_de_retalho.distrito AS distrito,
            ponto_de_retalho.concelho AS concelho,
            evento_reposicao.unidades AS unidades
      FROM produto, categoria, evento_reposicao, ponto_de_retalho,
planograma
      WHERE produto.cat = categoria.nome
            AND produto.ean = evento_reposicao.ean
            AND evento_reposicao.ean = planograma.ean
            AND planograma.loc = ponto_de_retalho.nome
);
```

## Desenvolvimento da aplicação

Os arquivos correspondentes a esta componente da avaliação estão disponíveis na pasta web/ (a subpasta templates contém os ficheiros html).

A versão de trabalho encontra-se em: http://web2.tecnico.ulisboa.pt/~ist195618/app.cgi/

A aplicação web tem 3 subcomponentes: Categorias, Retalhistas e Eventos. Ao abrir a aplicação, na diretoria raiz (‘/’), pode selecionar-se qual deles se pretende explorar.

O primeiro subcomponente, Categorias, encontra-se na subdiretoria ‘/categorias/’. Aí, está a implementação das alíneas a) e d) do enunciado.

**Alínea a) – inserir e remover categorias e sub-categorias:**

- Inserir categorias: carregando em “Inserir Categoria”, é-se redirecionado para o subendereço ‘/categorias/nova_cat’. Aí, especifica-se a categoria a adicionar (se é super ou simples e o seu nome);
- Remover categorias: bastando carregando em “APAGAR”, junto do nome de cada categoria (e confirmar essa decisão);
- Inserir sub-categorias: carregando em “Inserir Sub-Categoria”, é-se redirecionado para o subendereço ‘/categorias/nova_subcat’. Aí, escolhe-se a super-categoria desejada. Por fim, em ‘/categorias/nova_subcat/super’, seleciona-se qual a sub-categoria a associar;
- Remover sub-categorias: carregando em “Remover sub-categoria”, é-se redirecionado para o subendereço ‘/categorias/remo_subcat’. Aí, de todas as relações de super a sub-categoria, escolhe-se qual se quer remover.

**Alínea d) – listar todas as sub-categorias de uma super-categoria, a todos os níveis de profundidade:**

- Carregando em “Listar sub-categorias”, é-se redirecionado para o subendereço ‘/categorias/list_subcat’. Aí, seleciona-se a super-categoria desejada.

O segundo componente, Retalhistas, encontra-se na subdiretoria ‘/retalhistas/’. Aí, está a implementação da alínea b) do enunciado.

**Alínea b) – inserir e remover um retalhista, com todas as suas responsabilidades de reposições de produtos, garantindo que esta operação seja atómica:**

- Inserir um retalhista: carregando em “Inserir Retalhista”, é-se redirecionado para o subendereço ‘/retalhistas/novo_ret’. Aí, especifica-se o retalhista a adicionar (dando o seu TIN e nome);
- Remover um retalhista com todas as suas responsabilidades de reposições de produtos, garantindo que a operação é atómica: carregando em “Apagar”, ao lado de cada retalhista, é-se redirecionado para o subendereço ‘/retalhistas/remo_ret’. Aí, confirma-se a remoção do mesmo.

- **Nota**: na implementação escolhida, não é possível remover um retalhista se ele tiver efetuado algum evento de reposição (devido ao mesmo referenciar o retalhista – foreign key). Optou-se por essa via, e não por remover também os eventos, uma vez que se interpretou que um evento de reposição, ao ter ocorrido, não pode ser desfeito.

O terceiro componente, Eventos, encontra-se na subdiretoria '/eventos/'. Aí, está a implementação da alínea c) do enunciado.

**Alínea c) – listar todos os eventos de reposição de uma IVM, apresentando o número de unidades repostas por categoria de produto:**

- Ao entrar no endereço, escolhe-se qual a IVM cujos eventos de reposição se querem listar. Ao carregar em "Submit" aparece a informação desejada: os respetivos eventos de reposição, por categoria de produto (com o número de unidades repostas por cada).

Em todas as alíneas do enunciado procurou-se minimizar as vulnerabilidades a SQL injection. Reduziram-se as caixas de texto ao mínimo possível e, no código, todos os parâmetros são passados aos querys cuidadosamente (sem concatenação de strings).

## Consultas OLAP

[omitidas do relatório por permissão do enunciado, porém encontram-se no ficheiro analytics.sql]

## Índices

**7.1)** A tabela "retalhista" é completamente percorrida devido ao SELECT, no entanto, o mesmo não acontece para a tabela "responsavel_por" onde é feita uma procura pelas suas linhas, podendo ser beneficiada com a criação de um índice. Executando o comando "explain analyze" sobre a query pretendida, concluímos que temos uma procura sequencial à tabela "responsavel_por" e confirmamos que esta pode ser optimizada criando um índice sobre o campo "tin" e o campo "nome_cat".

```
create index tin_index on table responsavel_por(tin, nome_cat);
```

**7.2)** Executando o commando "explain analyze", concluímos que um index sobre a descrição do produto optimiza a query. Do mesmo modo, também poderá ser criado um índice para a categoria sendo que é feita uma comparação de igualdade dependente desta. Adicionalmente, será também benéfico a criação de um índice sobre o nome da tabela "tem_categoria" pois a eficiência do group by está dependente desta.

```
create index index_product on table produto(cat, descr);
create index_has_category on table tem_categoria(nome);
```