TÉCNICO LISBOA

Projeto BD - Parte 3

Grupo 164

Turno BD2L06

Prof. Sofia Maria Pais Cerqueira

```
$ SELECT nome, número, percentagem, esforço FROM Grupos WHERE id_grupo = 164;

       nome       | número | percentagem | esforço
------------------+--------+-------------+---------
 João Silva       | 99255  | "29%"       | "26h"
 Pedro Chaparro   | 99298  | "43%"       | "36h"
 Ricardo Almeida  | 99317  | "28%"       | "25h"
```

2021/2022

LEIC-A

1. Base de Dados

Esquema da BD

Por motivos de falta de espaço e seguindo a sugestão da prof. Sofia, apenas colocamos a referência para o ficheiro que contém o código. O esquema da BD (tais como as instruções para preencher a BD) encontra-se no ficheiro populate.sql, até à linha 134.

Nota 1: consideramos o atributo *ean* como *numeric*, pois tem exatamente 13 dígitos e portanto não pode ser *integer* (máx. 10 dígitos);

Nota 2: na tabela ***prateleira***, considerámos que a altura vem em centímetros.

2. Restrições de Integridade

À semelhança do ponto anterior, por motivos de falta de espaço, as restrições de integridade encontram-se no ficheiro ICs.sql.

Nota 1: Não foi tida em consideração a otimização na criação destas ICs.

Nota 2: Consideramos que uma prateleira não pode aceitar um produto cuja categoria é uma super categoria da categoria da prateleira, pois não faz sentido no mundo real.

3. SQL

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

SELECT s.nome
FROM (
    SELECT r.nome, COUNT(DISTINCT rp.nome_cat)
    FROM retalhista as r, responsavel_por as rp
    WHERE r.tin = rp.tin
    GROUP BY r.nome, rp.tin
) as s
WHERE s.count >= ALL (
    SELECT MAX(s.count)
    FROM (
    SELECT r.nome, COUNT(DISTINCT rp.nome_cat)
```

```
    FROM retalhista as r, responsavel_por as rp
    WHERE r.tin = rp.tin
    GROUP BY r.nome, rp.tin
) as s);

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

SELECT s.nome
FROM (
    SELECT r.nome, COUNT(DISTINCT rp.nome_cat)
    FROM retalhista as r, responsavel_por as rp, categoria_simples as c
    WHERE r.tin = rp.tin
        AND rp.nome_cat = c.nome
    GROUP BY r.nome, rp.tin
) as s
WHERE s.count >= ALL (
    SELECT COUNT (*)
    FROM categoria_simples
)

-- 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 (SELECT er.ean, COUNT(DISTINCT er.tin)
    FROM evento_reposicao as er
    GROUP BY er.ean
) as s
WHERE s.count = 1;
```

Nota 1: Interpretámos "mais sucinta" como a query ser o "mais perceptível".

4. Vistas

```
CREATE VIEW vendas(ean, cat, ano, trimeste, mes, dia_mes, dia_semana,
distrito, concelho, unidades)
AS
SELECT er.ean, p.cat, EXTRACT(YEAR FROM er.instante)::INTEGER,
EXTRACT(QUARTER FROM er.instante)::INTEGER,
    EXTRACT(MONTH FROM er.instante)::INTEGER, EXTRACT(DAY FROM
er.instante)::INTEGER,
    EXTRACT(DOW FROM er.instante)::INTEGER, pr.distrito, pr.concelho,
er.unidades
FROM evento_reposicao as er, produto as p, instalada_em as ie,
ponto_de_retalho as pr
WHERE er.ean = p.ean
    AND er.num_serie = ie.num_serie
    AND ie.local_ = pr.nome;
```

Nota 1: As expressões `::INTEGER` servem para fazer *cast* do atributo que é do tipo *Double*, pois a função `make_date()` usada no ponto 6. necessita que os argumentos sejam do tipo *Integer*.

5. Aplicação Web

Arquitetura da aplicação web

A aplicação é baseada em *python3*, usando:

- A framework `Flask`: usada para criar a app;
- A biblioteca de python `psycopg2`: usada para interagir com a BD;
- Ficheiros de `HTML`: usados para mostrar e interagir com o utilizador.

A app está guardada na pasta *web* e nela existem 3 ficheiros:

- Diretório `static`: onde são armazenadas as imagens usadas na app;
- Diretório `templates`: onde são armazenados os ficheiros HTML usados na app;
- Ficheiro `projetoBD-2021_2022-Entrega3.cgi`: aplicação web.

A app garante a atomicidade das operações sobre a base de dados - quando os métodos de `psycopg2` interagem com tabelas da base de dados começam, implicitamente, uma transação que apenas é acabada quando é chamado o método `.commit()` .

A app também previne ataques de *SQL injection* pois o método `.execute()` de `psycopg2` é sempre usado, passando os argumentos da *query* como parâmetros desse método.

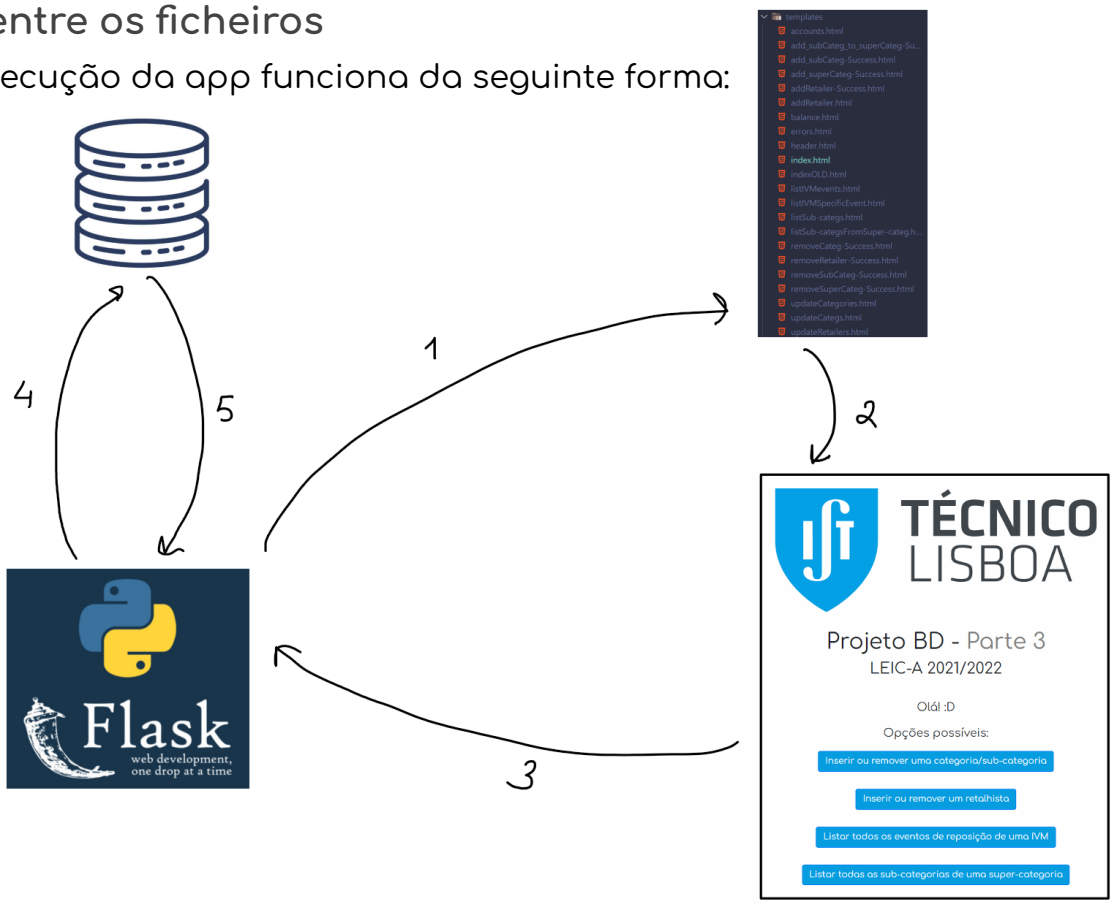
Link da versão de trabalho

A app encontra-se disponível no seguinte link:
http://web2.ist.utl.pt/ist199298/projetoBD-2021_2022-Entrega3.cgi/

A partir deste link, a app tem botões que permitem ser navegada sem ser necessário alterar o link manualmente na barra de pesquisa do navegador.

Relações entre os ficheiros

O flow de execução da app funciona da seguinte forma:



1. A app vai buscar ao diretório templates o ficheiro de entrada, `index.html`, e envia-o para o *browser*;
2. O *browser* renderiza o ficheiro, mostrando-o ao utilizador (indo buscar imagens ao diretório static, caso necessário);
3. O utilizador interage com o ficheiro `html`, clicando num dos botões (enviando então um pedido à app);

4. A app consulta/faz uma declaração à BD;
5. A BD responde com a consulta/output da declaração.

Depois, a interação repete-se: A app mostra um novo ficheiro `html`, o utilizador vê-o e escolhe uma opção, etc.

O passo 1. ocorre no chamamento da função `render_template()`, ex:

```
return render_template("updateCategories.html", cursor=cursor, FILENAME=FILENAME)
```

O passo 3. ocorre através de tags `<a>` `</a>` de `html`, que referenciam outras routes cobertas pelo `Flask`, ex:

```
<a href="update_categories" class="btn btn-primary">Inserir ou remover uma categoria/sub-categoria</a>
<br><br>
<a href="update_retailers" class="btn btn-primary">Inserir ou remover um retalhista</a>
<br><br>
<a href="list_IVM_events" class="btn btn-primary">Listar todos os eventos de reposição de uma IVM</a>
<br><br>
<a href="list_sub-categs" class="btn btn-primary">Listar todas as sub-categorias de uma super-categoria</a>
```

Os passos 4. e 5. Ocorrem através de chamadas a métodos da biblioteca `psycopg2`, ex:

```
dbConn = None
cursor = None
try:
    dbConn = psycopg2.connect(DB_CONNECTION_STRING)
    cursor = dbConn.cursor(cursor_factory=psycopg2.extras.DictCursor)

    query = "SELECT nome FROM categoria;"
    cursor.execute(query)
```

6. Consultas OLAP

Segundo o enunciado, o relatório não precisa incluir o componente OLAP.
Este está localizado no ficheiro *analytics.sql*.
Contudo, o grupo achou que deveria justificar algumas decisões.

Nota: À semelhança da 2ª entrega do projeto na parte da Álgebra Relacional e queries SQL onde foram usados os exemplos dados para construir a Álgebra/Query (ex: "*Para uma dada Categoria (e.g., "**Barras Energéticas**"), liste...*"), o grupo usou os exemplos dados para construir as queries:

- No caso de "i.e. entre duas datas", como o enunciado não foi explícito no intervalo, o grupo considerou um intervalo concreto:: 2020-07-09 até 2022-08-14;
- No caso de "i.e. 'Lisboa' ", considerámos o distrito como Lisboa.

6.1

O enunciado especifica que as vendas têm que ser apresentadas por "por dia da semana, por concelho e no total".
Com esta formulação da frase, o grupo acreditou que o pretendido era mostrar as vendas **exclusivamente** por dia da semana + **exclusivamente** por concelho (como se fossem dois `GROUP BY` separados) - Então, o grupo decidiu usar a instrução `GROUPING SETS` de forma a separar as agregações.

6.2

O grupo achou que ambas as instruções `ROLL UP` e `CUBE` seriam escolhas legítimas para esta query.
Contudo, o grupo achou que usar a instrução `CUBE` seria mais interessante pois, por um lado, mostra mais informação, e por outro, a instrução `ROLL UP` faria mais sentido num contexto de hierarquia (Se, por exemplo, a query quisesse antes mostrar os artigos vendidos por ano, trimestre, mês e dia do mês).

7. Índices

Nota sobre o enunciado

Apesar de ser dito no enunciado que "Suponha que não existam índices nas tabelas, além daqueles implícitos ao declarar chaves primárias e estrangeiras.". Isso não é de todo verdade - o postgreSQL apenas cria índices para as PKs e atributos `UNIQUE`, ou seja, não cria índices para FKs. Portanto, consideramos ignorar essa linha e criar índices para as FKs quando achássemos necessário.

7.1

```
CREATE INDEX responsavel_por_tin_idx ON responsavel_por USING HASH(tin);
CREATE INDEX responsavel_por_nome_cat_idx ON responsavel_por USING HASH
(nome_cat);
```

Nesta query, existem duas condições:

- `R.tin = P.tin` - é feito um JOIN que relaciona a PK da tabela ***retalhista*** com uma FK da tabela ***responsavel_por***. Então, faz sentido ter um índice de hash (visto se tratar de uma igualdade) sobre ***responsavel_por*** em *tin*.
- `P.nome_cat = 'Frutos'` - é feita uma comparação de igualdade. Para este tipo de comparação, faz sentido ter um índice também de Hash.

Então, devem-se criar índices hash *sobre a tabela* ***responsavel_por*** em *tin* e sobre a tabela ***produto*** em *nome_cat*.

7.2

```
CREATE INDEX produto_cat_idx ON produto USING HASH (cat);
CREATE INDEX produto_descr_idx ON produto(descr);
```

Nesta query, também existem duas condições:

- `P.cat = T.nome` - é feito um JOIN. Esta condição, à semelhança da anterior em 7.1, implica a criação de índices pois relaciona FKs das tabelas ***produto*** e ***tem_categoria***. Portanto, devem-se criar índices sobre ***produto*** em *cat* e sobre ***tem_categoria*** em *nome*, ambos do tipo hash pois especifica-se uma igualdade.
- `P.descr LIKE 'A%'` - é feita, indiretamente, uma comparação de intervalo - procuram-se todas as descrições entre 'Aa...' e 'Az...'. Como sabemos o início da palavra (ao contrário de, por exemplo, `LIKE '%A%'`), devemos então utilizar um índice Btree (devido a ser um intervalo) sobre ***P****.descr*.

Para além disso, é feita uma função de agregação, `count(T.ean)`, juntamente com um `GROUP BY`, que incidem sobre a tabela ***tem_categoria***. Esta tabela então também beneficia de um índice em (*nome, ean*) do tipo Btree, devido ao `GROUP BY`.
Este novo índice invalida então o índice de ***T*** apenas em *nome*.
Contudo, como é criada uma PK para a tabela ***tem_categoria*** com as colunas *ean* e *nome,* o índice já se encontra criado (por causa da criação feita pelo postgreSQL aquando da criação da tabela), ou seja, não é necessário criar um novo índice para esta tabela.

Finalmente, faz então sentido criar dois índices na tabela ***produto***, um índice Btree em ***P****.descr* e um índice Hash em ***P****.cat*.