TÉCNICO LISBOA

Bases de Dados

Projeto - E3

Grupo 55 - Turno L10

Professor Flávio Martins

Aluno	Esforço (horas)
Diogo Borges (99203)	10 horas (33%)
Francisco Lopes (99220)	10 horas (33%)
Nuno Martins (99292)	10 horas (33%)

Ano lectivo 2021/2022

Introdução

No presente relatório pretendemos demonstrar as nossas respostas relativas aos pontos mencionados no enunciado 3, das quais se destacam a criação do Esquema e Carregamento do mesmo na nossa Base de Dados, desenvolvimento de Restrições de Integridade, desenvolvimento de multiplas consultas SQL, OLAP e criação de VIEW, assim como a criação de um protótipo de aplicação web, com uso da framework de Python 'Flask'. De seguida apresentamos as respostas por ordem do seu aparecimento no enunciado.

1. Base de Dados

Esquema e carregamento da base de dados encontra-se no ficheiro anexado *populate.sql*

2. Restrições de Integridade

Restrições de Integridade com as extensões procedimentais SQL (Stored procedures e Triggers) no esquema de base de dados definido no ponto anterior encontra-se no ficheiro anexado *ICs.sql*

3. SQL

As consultas SQL e as 4 questões que respetivamente pretendem resolver encontram-se no ficheiro anexado *queries.sql* e são as seguintes:

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

```
SELECT x.retailer_name FROM (SELECT retailer_name, COUNT(*) AS count  FROM responsible_for INNER JOIN retailer ON (retailer.retailer_tin=responsible_for.retailer_tin)  GROUP BY retailer_name) AS x WHERE x.count = (SELECT MAX(x.count) FROM (SELECT retailer_name, COUNT(*) AS count  FROM responsible_for INNER JOIN retailer ON (retailer.retailer_tin=responsible_for.retailer_tin)  GROUP BY retailer_name) AS x);
```

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

```
SELECT DISTINCT retailer_name FROM retailer INNER JOIN (SELECT retailer_tin FROM responsible_for INNER JOIN (SELECT simple_category_name FROM simple_category INNER JOIN category ON (simple_category.simple_category_name = category.category_name)) AS x ON (responsible_for.category_name = x.simple_category_name)) AS y ON (retailer.retailer_tin = y.retailer_tin);
```

3. Quais os produtos (ean) que nunca foram repostos?

SELECT product_ean FROM product WHERE product_ean NOT IN (SELECT product_ean FROM replenish_event);

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

SELECT y.product_ean FROM (SELECT x.product_ean, COUNT(*) AS count FROM (SELECT DISTINCT product_ean,retailer_tin FROM replenish_event) AS x GROUP BY x.product_ean) AS y WHERE y.count = 1;

4. Vistas

A vista criada que resume as informações masi importantes sobre as vendas, combinando informações de diferentes tabelas do modelo e com o esquema – Vendas(ean, cat, ano, trimestre, dia_mes, dia_semana, distrito, concelho, unidades) – encontra-se no ficheiro anexado *view.sql* e é a seguinte:

```
DROP VIEW sales;

create or replace function GET_TRIMESTRE(timestamp)
returns integer language sql immutable as $$
    select (extract('month' from $1::date-9)::int-1)/3+1
$$;

CREATE VIEW sales
AS
SELECT DISTINCT
replenish_event.product_ean ean,
category_name cat,
TO_CHAR(replenish_event_instant::timestamp,'YYYY') ano,
GET_TRIMESTRE(replenish_event_instant::timestamp) trimestre,
TO_CHAR(replenish_event_instant::timestamp,'DD') dia_mes,
TO_CHAR(replenish_event_instant::timestamp,'Day') dia_semana,
retail_point_district distrito,retail_point_county concelho,
replenish_event_units unidades
FROM replenish_event
INNER JOIN (SELECT category_name, product.product_ean,
retail_point_district,retail_point_county FROM product
INNER JOIN (SELECT product_ean,retail_point_district,retail_point_county FROM
retail_point
INNER JOIN (SELECT * FROM installed_at
INNER JOIN (SELECT product.product_ean,ivm_serial_number FROM replenish_event
INNER JOIN product ON (replenish_event.product_ean = product.product_ean)) AS x ON
(installed_at.ivm_serial_number = x.ivm_serial_number))
AS y ON (retail_point. retail_point_name = y.retail_point_name)) AS w ON
(product.product_ean = w.product_ean)) AS a ON (replenish_event.product_ean = a.
product_ean);
```

Nesta criação de View, foi usada uma função auxiliar 'GET_TRIMESTRE(timestamp)' que recebe uma timestamp e devolve o numero do trimestre (1,2,3 ou 4).

5. Desenvolvimento da Aplicação

LINK para a aplicação: **https://web2.tecnico.ulisboa.pt/ist199292/proj.cgi/**

Estrutura de ficheiros da aplicação:

```
~
|
-web
|    |-templates
|    |    |-add-category.html
|    |    |-add-retailer.html
|    |    |-category.html
|    |    |-delete-category.html
|    |    |-delete-retailer.html
|    |    |-home.html
|    |    |-ivm.html
|    |    |-retailer.html
|    |    |-subcategory.html
|    |- proj.cgi
```

A aplicação foi criada com a framework de Python 'Flask', em que conecta à base de dados e dá 'render' das páginas HTML que se econtram armazenadas na pasta 'templates'. A página inicial é a 'home.html' onde temos 4 opções, opções essas que são as descritas no enunciado na secção 5 e são apresentadas em pormenor de seguida:

a) **Inserir e remover categorias e sub-categorias:**

Para realizar esta ação, foi criado um botão dentro da 'home.html' que ao ser clicado redireciona o utilizador para a página 'category.html'. Nela é possivel fazer a leitura das categorias armazenadas na nossa base de dados, assim como uma opção para apagar e acrescentar nova categoria. Ao apagar, a nossa aplicação envia várias queries que apagam essa categoria, assim como todas as 'rows' onde essa categoria era usada, nomeadamente no planograma, no evento de reposição, nos produtos, no responsável_por, entre outros, e posteriormente redireciona para a página 'delete-category.html' onde o utilizador vê uma mensagem de sucesso e uma opção para voltar atrás. No caso da opção para adicionar nova categoria, o utilizador ao preencher e submeter o nome da categoria, o input é validado pela nossa aplicação para evitar ataques por injeção de SQL e posteriormente é criada uma categoria com o nome passado por input, redirecionando o utilizador para a página 'add-category.html' onde vê uma mensagem de sucesso e um botão para regressar à pagina anterior. Após regressar à página anterior podemos ver as alterações, tanto de categorias apagadas como de categorias adicionadas.

b) Inserir e remover um retalhista, com todos os seus produtos, garantindo que esta operação seja atómica:

Para realizar esta ação, foi criado um botão dentro da 'home.html' que ao ser clicado redireciona o utilizador para a página 'retailer.html'. Nela é possivel fazer a leitura dos TIN dos retalhistas, assim como o seu nome armazenados ambos na nossa base de dados, assim como uma opção para apagar e acrescentar novo retalhista (com TIN e nome). Ao apagar um retalhista, a nossa aplicação envia várias queries que apagam esse retalhista, assim como todas as 'rows' onde esse retalhista era usado, nomeadamente no evento de resposição, no 'responsible_for' e no próprio retailer e posteriormente redireciona para a página 'delete-category.html' onde o utilizador vê uma mensagem de sucesso e uma opção para voltar atrás. No caso da opção para adicionar nova retalhista, o utilizador ao preencher e submeter o TIN e nome do retalhista, o input é validado pela nossa aplicação para evitar ataques por injeção de SQL e posteriormente é criado um novo retalhista com o TIN e nome passados por input, redirecionando o utilizador para a página 'delete-retailer.html' onde vê uma mensagem de sucesso e um botão para regressar à pagina anterior. Após regressar à página anterior podemos ver as alterações, tanto de retalhistas apagados como de retalhistas adicionados.

c) Listar todos os eventos de reposição de uma IVM, apresentando o número de unidades repostas por categoria de produto:

Para realizar esta ação, foi criado um input field, onde o utilizador insere o serial_number da IVM sobre a qual pretende listar todos os eventos de reposição e posteriormente é redirecionado para a página 'ivm.html' onde consegue visualizar uma tabela com o número de unidades repostas por categoria de produto. A query usada para listar as informações pedidas é a seguinte:

```
SELECT x.category_name, SUM(x.replenish_event_units) AS total_units FROM
(SELECT category_name,replenish_event_units FROM product INNER JOIN
replenish_event ON (product.product_ean = replenish_event.product_ean)
ivm_serial_number = {ivm_serial}) AS x GROUP BY x.category_name;
```

Nota: {ivm_serial} é o serial_number passado como input pelo utilizador.

d) Listar todas as sub-categorias de uma super-categoria, a todos os níveis de profundidade.

Para realizar esta ação, foi criado um input field, onde o utilizador insere o nome da categoria sobre a qual pretende listar todas as suas subcategorias, a todos os niveis de profundidade e posteriormente é redirecionado para a página 'subcategory.html' onde consegue visualizar uma tabela com o nome de todas as subcategorias a todos os niveis de profundidade. A query usada para listar as informações pedidas de forma recursiva é a seguinte:

```
WITH RECURSIVE New AS (SELECT category_name AS c FROM has_other WHERE
super_category_name = '{supercategory_name}' UNION ALL SELECT category_name FROM
New,has_other WHERE New.c = has_other.super_category_name) SELECT * FROM New;
```

Nota: {supercategory_name} é o nome da categoria passado como input pelo utilizador.

6. Consultas OLAP

As consultas OLAP usando a vista desenvolvida para a questão 4, encontram-se no ficheiro anexado *analytics.sql* e estão descritas de seguida:

a) num dado período (i.e. entre duas datas), por dia da semana, por concelho e no total

```
SELECT s.ano,s.dia_semana,s.concelho, COUNT(s.ean) FROM sales AS s
WHERE ano::decimal  BETWEEN 2005 AND 2022
GROUP BY CUBE(s.ano, s.dia_semana, s.concelho);
```

b) num dado distrito (i.e. “Lisboa”), por concelho, categoria, dia da semana e no total

```
SELECT s.distrito,s.concelho,s.cat,s.dia_semana,COUNT(s.ean) FROM sales AS s
WHERE s.distrito = 'setubal'
GROUP BY CUBE(s.distrito,s.concelho,s.cat,s.dia_semana);
```

Nota: para exemplificar as consultas OLAP foi usado o período entre 2005 e 2022 no caso da consulta a) e o distrito ‘setubal’ para a consulta b).

7. Índices

7.1

CREATE INDEX Index_P ON P(cat_name, tin);

CREATE INDEX Index_tin ON R(tin)

No primeiro exemplo foram usadas duas chaves de procura compostas de modo a otimizar o tempo de execução. Ao indexar o cat_name, facilita desde já a restrição de valores para P.nome_cat = ‘Frutos’.

De seguida indexando p.tin e também criando outro índice para R, indexando também as tin facilitando o match de R.tin = P.tin.

7.2

CREATE INDEX Index_name ON T(nome)

CREATE INDEX Index_cat ON P(cat)

Neste exemplo foram usadas também duas chaves de procura. Um índice para indexar T.nome; outro índice para indexar P.cat; Facilitando assim o match de p.cat = T.nome.