Instituto Superior Técnico

LEIC 2021/2022

Projeto BD - Parte 3

Professor Francisco Regateiro

Grupo nº 93, turno B2L09

Nome, número e aproveitamento:

- Gonçalo Azevedo – nº 93075 – 33% (18 horas)
- Ivan Fortes – nº 99085 – 33% (18 horas)
- Paulo Almeida – nº 98959 – 33% (18 horas)

Ficheiro de criação da Base de dados (schema.sql)

```
DROP TABLE IF EXISTS category CASCADE;

DROP TABLE IF EXISTS simple_category CASCADE;

DROP TABLE IF EXISTS super_category CASCADE;

DROP TABLE IF EXISTS has_other CASCADE;

DROP TABLE IF EXISTS product CASCADE;

DROP TABLE IF EXISTS has_category CASCADE;

DROP TABLE IF EXISTS ivm CASCADE;

DROP TABLE IF EXISTS point_of_retail CASCADE;

DROP TABLE IF EXISTS installed_at CASCADE;

DROP TABLE IF EXISTS shelf CASCADE;

DROP TABLE IF EXISTS planogram CASCADE;

DROP TABLE IF EXISTS retailer CASCADE;

DROP TABLE IF EXISTS responsible_for CASCADE;

DROP TABLE IF EXISTS replenishment_event CASCADE;


CREATE TABLE IF NOT EXISTS category (

category_name        varchar(80)          NOT NULL UNIQUE,

PRIMARY KEY(category_name));


CREATE TABLE IF NOT EXISTS simple_category (

category_name  varchar(80)  NOT NULL UNIQUE,

PRIMARY KEY(category_name),

CONSTRAINT fk_simple_category FOREIGN KEY(category_name) REFERENCES category(category_name) ON DELETE
CASCADE);


CREATE TABLE IF NOT EXISTS super_category (

 category_name  varchar(80)  NOT NULL UNIQUE,

 PRIMARY KEY(category_name),

 CONSTRAINT fk_super_category  FOREIGN KEY(category_name) REFERENCES         category(category_name) ON
DELETE CASCADE);


CREATE TABLE IF NOT EXISTS has_other (

 super_category   varchar(80)              NOT NULL,

 category             varchar(80)     NOT NULL,
```

```
PRIMARY KEY(category),

CONSTRAINT fk_other_superFOREIGN KEY(super_category)        REFERENCES super_category(category_name)
ON DELETE CASCADE,

CONSTRAINT fk_other_category FOREIGN KEY(category)  REFERENCES category(category_name) ON DELETE
CASCADE,

CHECK(super_category <> category));
```
(RI-RE2)

```
CREATE TABLE IF NOT EXISTS product (

ean          NUMERIC(13,0      NOT NULL UNIQUE,

category      varchar(80)     NOT NULL,

description    varchar(80)    NOT NULL,

PRIMARY KEY(ean),

CONSTRAINT fk_product_category       FOREIGN KEY(category)       REFERENCES  category(category_name) ON
DELETE CASCADE);

CREATE TABLE IF NOT EXISTS has_category (

ean                          NUMERIC(13,0)      NOT NULL,

category_name                varchar(80)                NOT NULL,

CONSTRAINT fk_has_product            FOREIGN KEY(ean)  REFERENCES product(ean),

CONSTRAINT fk_has_category           FOREIGN KEY(category_name)          REFERENCES
category(category_name) ON DELETE CASCADE);

CREATE TABLE IF NOT EXISTS ivm (

serial_number       varchar(80)          NOT NULL,

manufacturer        varchar(80)          NOT NULL,

PRIMARY KEY(serial_number, manufacturer));

CREATE TABLE IF NOT EXISTS point_of_retail(

location_name               varchar(80)          NOT NULL UNIQUE,

location_district     varchar(80)          NOT NULL,

location_county     varchar(80)          NOT NULL,

PRIMARY KEY(location_name));

CREATE TABLE IF NOT EXISTS installed_at(

serial_number               varchar(80)          NOT NULL,

manufacturer                varchar(80)          NOT NULL,

local                       varchar(80)       NOT NULL,
```

```
PRIMARY KEY(serial_number, manufacturer),

CONSTRAINT fk_installed_serial FOREIGN KEY(serial_number, manufacturer) REFERENCES ivm(serial_number, manufacturer) ON DELETE CASCADE,

CONSTRAINT fk_installed_local FOREIGN KEY(local)
        REFERENCES point_of_retail(location_name) ON DELETE CASCADE);
```

CREATE TABLE IF NOT EXISTS shelf (

```
shelf_number    INT             NOT NULL,

serial_number   varchar(80)     NOT NULL,

manufacturer    varchar(80)     NOT NULL,

height          INT             NOT NULL,

category_name   varchar(80)     NOT NULL,

UNIQUE(shelf_number, serial_number, manufacturer),

PRIMARY KEY(shelf_number, serial_number, manufacturer),

CONSTRAINT fk_shelf_category FOREIGN KEY(category_name) REFERENCES category(category_name) ON DELETE CASCADE,

CONSTRAINT fk_shelf_ivm_serial FOREIGN KEY(serial_number, manufacturer) REFERENCES ivm(serial_number, manufacturer) ON DELETE CASCADE);
```

CREATE TABLE IF NOT EXISTS planogram (

```
ean             NUMERIC(13,0)       NOT NULL,

shelf_number    INT                 NOT NULL,

serial_number   varchar(80)         NOT NULL,

manufacturer    varchar(80)         NOT NULL,

faces           INT                 NOT NULL,

units           INT                 NOT NULL,

loc             varchar(80)         NOT NULL,

PRIMARY KEY(shelf_number, ean, serial_number, manufacturer),

CONSTRAINT fk_planogram_product_ean FOREIGN KEY(ean) REFERENCES product(ean) ON DELETE CASCADE,

CONSTRAINT fk_planogram_shelf_number FOREIGN KEY(shelf_number, serial_number, manufacturer)
REFERENCES shelf(shelf_number, serial_number, manufacturer) ON DELETE CASCADE);
```

CREATE TABLE IF NOT EXISTS retailer (

```
        tin             INT             NOT NULL UNIQUE,

        retailer_name   varchar(80)     NOT NULL UNIQUE,    (RI-RE7)

        PRIMARY KEY(tin));
```

CREATE TABLE IF NOT EXISTS responsible_for (

```
        category_name   varchar(80)     NOT NULL,
```

```
    tin                 INT                 NOT NULL,
    serial_number       varchar(80)         NOT NULL,
    manufacturer        varchar(80)         NOT NULL,
    PRIMARY KEY(serial_number, manufacturer),
    CONSTRAINT fk_responsible_category FOREIGN KEY(category_name) REFERENCES
    category(category_name) ON DELETE CASCADE,
    CONSTRAINT fk_responsible_retailer  FOREIGN KEY(tin) REFERENCES retailer(tin)
    ON DELETE CASCADE,
    CONSTRAINT fk_responsible_ivm_serial FOREIGN KEY(serial_number, manufacturer)  REFERENCES
    ivm(serial_number, manufacturer) ON DELETE CASCADE);
```

```
CREATE TABLE replenishment_event (
ean                         NUMERIC(13,0)       NOT NULL,
shelf_number                INT                 NOT NULL,
serial_number               varchar(80)         NOT NULL,
manufacturer                varchar(80)         NOT NULL,
instant                     TIMESTAMP           NOT NULL,
replenished_units           INT                 NOT NULL,
tin                         INT                 NOT NULL,
PRIMARY KEY(ean, shelf_number, serial_number, manufacturer, instant),
CONSTRAINT fk_replenishment_planogram  FOREIGN KEY(ean, shelf_number, serial_number, manufacturer)
        REFERENCES planogram(ean, shelf_number, serial_number, manufacturer) ON DELETE CASCADE,
CONSTRAINT fk_replenishment_retailer FOREIGN KEY(tin) REFERENCES retailer(tin) ON DELETE CASCADE);
```

- Foram implementadas as restrições RI-RE2 e RI-RE7 do esquema relacional no código do schema.sql.
- Operações como o CASCADE e o ON DELETE CASCADE foram utilizadas para criar e atualizar tabelas dependentes de outras tabelas.

Restrições de Integridade

```
DROP TRIGGER IF EXISTS super_category_child ON has_other;

DROP TRIGGER IF EXISTS planogram_limit_units ON replenishment_event;

DROP TRIGGER IF EXISTS shelf_category ON replenishment_event;


-- IC-1, uma categoria não pode conter-se a si mesma.

CREATE OR REPLACE FUNCTION chk_super_category_child()

RETURNS TRIGGER AS

$$

BEGIN
```

```
        IF NEW.super_category = NEW.category THEN
                RAISE EXCEPTION 'Category "%" cannot contain itself.', NEW.super_category;
        END IF;
        RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER super_category_child
BEFORE UPDATE OR INSERT ON has_other
FOR EACH ROW EXECUTE PROCEDURE chk_super_category_child();
```

-- IC-2, O nº de unidades de um evento de reposição não pode exceder o nº descrito de unidades no planograma.

```
CREATE OR REPLACE FUNCTION chk_planogram_units_limit()
RETURNS TRIGGER AS
$$
DECLARE limit_units INT;
BEGIN
        SELECT units INTO limit_units
        FROM planogram p
        WHERE p.ean = NEW.ean AND p.shelf_number = NEW.shelf_number AND p.serial_number = NEW.serial_number AND p.manufacturer = NEW.manufacturer;
        IF NEW.replenished_units > limit_units THEN
                RAISE EXCEPTION 'Replenished units, "%", exceed the max limit of units, "%", specified in the planogram.', NEW.replenished_units, limit_units;
        END IF;
        RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER planogram_limit_units
BEFORE INSERT ON replenishment_event
FOR EACH ROW EXECUTE PROCEDURE chk_planogram_units_limit();
```

-- IC-3, Um produto só pode ser reposto numa prateleira destinada a uma das suas categorias.

```
CREATE OR REPLACE FUNCTION chk_shelf_category()
RETURNS TRIGGER AS
$$
```

```
DECLARE shelf_category VARCHAR(80);
BEGIN
        SELECT category_name INTO shelf_category
        FROM shelf s
        WHERE s.shelf_number = NEW.shelf_number AND s.serial_number = NEW.serial_number AND s.manufacturer = NEW.manufacturer;
        IF shelf_category NOT IN (
                SELECT category_name
                FROM has_category p
                WHERE p.ean = NEW.ean) THEN
                RAISE EXCEPTION 'Shelf is not destined for the categories of product with ean"%".', NEW.ean;
        END IF;
        RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER shelf_category
BEFORE INSERT ON replenishment_event
FOR EACH ROW EXECUTE PROCEDURE chk_shelf_category();
```

Queries

```
-- Query 1 - Qual o retalhista responsável pelo maior nº de categorias?
SELECT retailer_name
FROM responsible_for NATURAL JOIN retailer
GROUP BY retailer_name
HAVING COUNT(DISTINCT category_name) >= ALL (
        SELECT COUNT(DISTINCT category_name)
        FROM responsible_for
        GROUP BY tin);
-- Query 2 - Qual o nome dos retalhistas que são responsáveis por todas as categorias simples?
-- Division using EXCEPT and correlated queries
SELECT retailer_name
FROM retailer r
WHERE NOT EXISTS (
        SELECT category_name
        FROM simple_category
        EXCEPT
```

```
        SELECT DISTINCT category_name
        FROM responsible_for NATURAL JOIN retailer RR
        WHERE RR.retailer_name = r.retailer_name);
```

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

```
SELECT ean
FROM product
EXCEPT
SELECT DISTINCT ean
FROM replenishment_event;
```

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

```
SELECT ean
FROM replenishment_event
GROUP BY ean
HAVING COUNT(DISTINCT tin) = 1;
```

Observação:

Em relação à segunda query, dado o atual esquema relacional apenas será possível existir um retalhista responsável por todas as categorias simples se existirem, no mínimo, tantas IVMs quantas categorias simples e, para esse caso concreto, o mesmo retalhista aparecer em todas as entradas da tabela responsible_for associado em cada registo a uma categoria simples diferente. Isto acontece devido à chave principal da tabela responsible_for ser chave estrangeira da chave principal da tabela IVM, isto é, apenas se pode atribuir responsabilidade de uma categoria a um retalhista por cada IVM existente.

Vista (view.sql)

```
DROP VIEW IF EXISTS sales;
CREATE VIEW IF NOT EXISTS sales
AS (
        SELECT ean, category_name, EXTRACT(YEAR FROM instant) AS year, EXTRACT(QUARTER FROM instant) AS quarter, EXTRACT(MONTH FROM instant) AS month, EXTRACT(DAY FROM instant) AS day_month, EXTRACT(DOW FROM instant) AS day_week, location_district, location_county, replenished_units
        FROM (
        SELECT ean, category_name, instant, replenished_units, location_district, location_county
        FROM replenishment_event NATURAL JOIN responsible_for NATURAL JOIN installed_at NATURAL JOIN point_of_retail) AS base_table);
```

- Para criar a vista, foi utilizada a função **EXTRACT** para retirar os valores específicos da variável **instant** (do tipo timestamp). O segundo **FROM** foi utilizado para criar uma tabela que relacionava as variaveis descritas no segundo **SELECT**, e de seguida o primeiro **SELECT** retirava a informação de **instant** para outras novas variáveis (year, quarter, month, day_month e day_week).

Arquitetura da aplicação web

- É possível aceder à aplicação web através do link: http://web2.tecnico.ulisboa.pt/ist198959/app.cgi/

- Acedendo ao link, é apresentado uma página onde temos um menu inicial, sendo possível consultar todas as tabelas da base de dados como também realizar outras operações. As operações apresentadas permitem adicionar e remover categorias e retalhistas, como também consultar os eventos da IVM desejada ou consultar todas as subcategorias de uma categoria.

- Nas operações referentes a uma IVM e categoria são pedidas o nº de série da IVM e o nome da categoria (respetivamente), sendo que no caso de adicionar retalhista é pedido o seu nome e o seu TIN, e no remover retalhista apenas o seu nome (não foi escolhido o TIN pois a remoção seria mais simples por nome).

- De forma a apresentar toda esta informação e operações, além da utilização dos ficheiros schema.sql e populate.sql utilizados para criar e encher a base de dados, foram também utilizados um ficheiro python (app.cgi) em conjunto com uma pasta template contendo 24 ficheiros html para obter as tabelas da base de dados e representar as mesmas em diversas páginas. Na root da aplicação foi utilizado o ficheiro index.html para servir de menu para outras funções do ficheiro python, onde estas funções utilizam outros ficheiros html.

- A aplicação web foi feita para prevenir ataques por SQL INJECTION através das funções que utiliza.

Índices

7.1 -

```
SELECT DISTINCT R.nome

FROM retalhista R, responsavel_por P

WHERE R.tin = P.tin and P. nome_cat = 'Frutos'
```

No sentido de agilizar a execução de cada consulta faria sentido criar os seguintes índices: Um do tipo Hash na tabela retalhista, para o atributo nome, uma vez que os duplicados são removidos(ou seja há um sorting).

Um do tipo Hash na tabela responsavel_por, para o atributo nome_cat. A escolha do tipo de índice deve se ao facto de os índices Hash serem os melhores para seleção por igualdade.

Existe um índice implícito para a chave primária tin em ambas as tabelas, logo não é necessária a criação de um suplementar para esse atributo.

7.2 -

```
SELECT T.nome, count(T.ean)

FROM produto P, tem_categoria T

WHERE p.cat = T.nome and P.desc like 'A%'

GROUP BY T.nome
```

Neste caso faria sentido a criação dos índices:

Um do tipo Bitmap na tabela produto, para os atributos cat e desc (por esta ordem, que está em função da seletividade dos atributos). A escolha do tipo de índice deve se ao facto de os índices Bitmap serem especialmente vantajosos em interrogações sobre vários atributos.

Um do tipo Tree na tabela tem_categoria, para os atributos nome e ean.