PROJETO BD - PARTE 3

Grupo 20 - Turno BD2L05

Professor: Francisco Regateiro

MEMBROS / CONTRIBUIÇÃO / ESFORÇO TOTAL

Valentim Santos	ist199343	33.3 %	32 horas
Tiago Santos	ist199333	33.3 %	32 horas
Yassir Yassin	ist1100611	33.3 %	32 horas

SQL
Restrições de Integridade

(RI-1) Uma categoria nao pode estar contida em si propria

```
CREATE OR REPLACE FUNCTION catNotInItself() RETURNS TRIGGER AS $$
   DECLARE
    superCat varchar(100);
   BEGIN
   superCat := NEW.super_categoria;

    -- If the super_category != category, then there is still a chance that there's
    -- a loop in the sub-categories, and so, we check if the super category
    -- is already apart of its sub-categories.
    IF superCat = NEW.categoria OR superCat IN (

      -- Recursively gets all the sub-categories of the super category.
      WITH RECURSIVE cte AS (
      SELECT categoria AS a
        FROM tem_outra
        WHERE super_categoria = NEW.categoria
      UNION ALL
      SELECT categoria
        FROM cte INNER JOIN tem_outra
        ON super_categoria = a
      )
      SELECT * FROM cte
      UNION ALL
        SELECT nome
        FROM categoria
          WHERE nome = NEW.categoria

    ) THEN
      RAISE EXCEPTION 'Uma categoria nao pode estar contida em si propria.';
      RETURN OLD;

    END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER catNotInItselfTrigger
  BEFORE INSERT OR UPDATE
  ON tem_outra
  FOR EACH ROW
  EXECUTE PROCEDURE catNotInItself();
```

(RI-4) O numero de unidades repostas num Evento de Reposicao nao pode exceder o numero de unidades especificado no Planograma.

```
CREATE OR REPLACE FUNCTION nRepoLessThanMax() RETURNS TRIGGER AS $$
  DECLARE
  un integer;
  BEGIN
    -- Stores in 'un' the maximum number of units that can be replaced,
    -- established by the Planogram.
    un := unidades FROM planograma WHERE planograma.ean = NEW.ean AND planograma.nro = NEW.nro;
    -- If the units trying to be replaced exceed said amount, an exception
    -- is thrown, NEW is returned and the entries are not inserted into the table.
    IF NEW.unidades > un THEN
      RAISE EXCEPTION 'O numero de unidades repostas num Evento de Reposicao nao pode exceder o numero de unidades especificado no Planograma.';
      RETURN NEW;
    END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER nRepoLessThanMaxTrigger
  BEFORE INSERT OR UPDATE
  ON evento_reposicao
  FOR EACH ROW
  EXECUTE PROCEDURE nRepoLessThanMax();
```

SQL

Restrições de Integridade (continuação)

(RI-5) Um Produto so pode ser reposto numa Prateleira que apresente (pelo menos) uma das Categorias desse produto.

```
CREATE OR REPLACE FUNCTION prateCategoria() RETURNS TRIGGER AS $$
  DECLARE
    prodCat varchar(100);
    prateCat varchar(100);
  BEGIN
    -- Stores in 'prodCat' the product's category .
    SELECT cat INTO STRICT prodCat
      FROM produto WHERE produto.ean = NEW.ean;

    -- Stores in 'prateCat' the shelf's category.
    SELECT nome INTO STRICT prateCat
      FROM prateleira WHERE prateleira.nro = NEW.nro AND prateleira.num_serie = NEW.num_serie;

      -- If the new product's category is different from the shelf's category,
      -- then there is a chance that the shelf holds a sub-category equal to
      -- the product's, and so, we recursively check for it.
      IF prodCat IS DISTINCT FROM prateCat AND prodCat NOT IN (

        -- Recursively gets all the sub-categories of the shelf's category.
        WITH RECURSIVE cte AS (
        SELECT categoria AS a
          FROM tem_outra
          WHERE super_categoria = prateCat
        UNION ALL
        SELECT categoria
          FROM cte INNER JOIN tem_outra
          ON super_categoria = a
        )
        SELECT * FROM cte
        UNION ALL
          SELECT nome
          FROM categoria
            WHERE nome = prateCat

      ) THEN
        RAISE EXCEPTION 'Um produto apenas pode ser reposto numa prateleira que apresente (pelo menos) uma das categorias dele mesmo.';
        RETURN NEW;
      END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER prateCategoriaTrigger
  BEFORE INSERT OR UPDATE
  ON evento_reposicao
  FOR EACH ROW
  EXECUTE PROCEDURE prateCategoria();
```

SQL

Schema

```
DROP TABLE IF EXISTS categoria CASCADE;
DROP TABLE IF EXISTS categoria_simples CASCADE;
DROP TABLE IF EXISTS super_categoria CASCADE;
DROP TABLE IF EXISTS tem_outra CASCADE;
DROP TABLE IF EXISTS produto CASCADE;
DROP TABLE IF EXISTS tem_categoria CASCADE;
DROP TABLE IF EXISTS IVM CASCADE;
DROP TABLE IF EXISTS ponto_de_retalho CASCADE;
DROP TABLE IF EXISTS instalada_em CASCADE;
DROP TABLE IF EXISTS prateleira CASCADE;
DROP TABLE IF EXISTS planograma CASCADE;
DROP TABLE IF EXISTS retalhista CASCADE;
DROP TABLE IF EXISTS responsavel_por CASCADE;
DROP TABLE IF EXISTS evento_reposicao CASCADE;
```

```
-- Table: categoria
CREATE TABLE categoria (
  nome VARCHAR(100),

  PRIMARY KEY (nome)
);


-- Table: categoria_simples
CREATE TABLE categoria_simples (
  nome VARCHAR(100),

  PRIMARY KEY(nome),
  FOREIGN KEY(nome) references categoria(nome)
);
```

SQL

Schema (continuação)

```
-- Table: super_categoria
REATE TABLE super_categoria (
  nome VARCHAR(100),

  PRIMARY KEY(nome),
  FOREIGN KEY(nome) references categoria(nome)
);


-- Table: tem_outra
CREATE TABLE tem_outra (
  super_categoria VARCHAR(100) NOT NULL,
  categoria VARCHAR(100),

  PRIMARY KEY(categoria),
  FOREIGN KEY(super_categoria) references super_categoria(nome),
  FOREIGN KEY(categoria) references categoria(nome)
);


-- Table: produto
CREATE TABLE produto (
  ean CHAR(13),
  cat VARCHAR(100) NOT NULL,
  descr VARCHAR(100),

  PRIMARY KEY(ean),
  FOREIGN KEY(cat) references categoria(nome)
);


-- Table: tem_categoria
CREATE TABLE tem_categoria (
  ean CHAR(13),
  nome VARCHAR(100),

  PRIMARY KEY(ean, nome),
  FOREIGN KEY(ean) references produto(ean),
  FOREIGN Key(nome) references categoria(nome)
);


-- Table: IVM
CREATE TABLE IVM (
  num_serie INTEGER,
  fabricante VARCHAR(100),

  PRIMARY KEY(num_serie, fabricante)
);


-- Table: ponto_de_retalho
CREATE TABLE ponto_de_retalho (
  nome VARCHAR(100),
  distrito VARCHAR(100) NOT NULL,
  concelho VARCHAR(100) NOT NULL,

  PRIMARY KEY(nome)
);


-- Table: instalada_em
CREATE TABLE instalada_em (
 num_serie INTEGER,
 fabricante VARCHAR(100),
 loc VARCHAR(100) NOT NULL,

 PRIMARY KEY(num_serie, fabricante),
 FOREIGN KEY(num_serie, fabricante) references IVM(num_serie,
fabricante),
 FOREIGN KEY(loc) references ponto_de_retalho(nome)
);
```

```
-- Table: prateleira
CREATE TABLE prateleira (
  nro INTEGER,
  num_serie INTEGER,
  fabricante VARCHAR(100),
  altura INTEGER NOT NULL,
  nome VARCHAR(100) NOT NULL,

  PRIMARY KEY(nro, num_serie, fabricante),
  FOREIGN KEY(num_serie, fabricante) references IVM(num_serie,
                                                    fabricante),
  FOREIGN KEY(nome) references categoria(nome)
);


-- Table: planograma
CREATE TABLE planograma (
  ean CHAR(13),
  nro INTEGER,
  num_serie INTEGER,
  fabricante VARCHAR(100),
  faces INTEGER NOT NULL,
  unidades INTEGER,
  loc VARCHAR(100) NOT NULL,

  PRIMARY KEY(ean, nro, num_serie, fabricante),
  FOREIGN KEY(ean) references produto(ean),
  FOREIGN KEY(nro, num_serie, fabricante) references
                          prateleira(nro, num_serie, fabricante)
);


-- Table: retalhista
CREATE TABLE retalhista (
  tin INTEGER,
  nome VARCHAR(100) NOT NULL UNIQUE,

  PRIMARY KEY(tin)
);


-- Table: responsavel_por
CREATE TABLE responsavel_por (
  nome_cat VARCHAR(100) NOT NULL,
  tin INTEGER NOT NULL,
  num_serie INTEGER,
  fabricante VARCHAR(100),

  PRIMARY KEY(num_serie, fabricante),
  FOREIGN KEY(num_serie, fabricante) references IVM(num_serie,
                                                    fabricante),
  FOREIGN KEY(tin) references retalhista(tin),
  FOREIGN KEY(nome_cat) references categoria(nome)
);


-- Table: evento_reposicao
CREATE TABLE evento_reposicao (
  ean CHAR(13),
  nro INTEGER,
  num_serie INTEGER,
  fabricante VARCHAR(100),
  instante timestamp,
  unidades INTEGER,
  tin INTEGER NOT NULL,

  PRIMARY KEY(ean, nro, num_serie, fabricante, instante),
  FOREIGN KEY(ean, nro, num_serie, fabricante) references
                    planograma(ean, nro, num_serie, fabricante),
  FOREIGN KEY(tin) references retalhista(tin)
);
```

SQL

Schema: Triggers

Para que as operações de remove em ambas as tabelas **categoria** e **retalhista** fossem possíveis foram criados diversos triggers, essencialmente funcionando como um "**ON DELETE CASCADE**".

```
-- Table: prateleira (Trigger)
CREATE OR REPLACE FUNCTION remove_shelf()
RETURNS TRIGGER AS
$$
BEGIN
  DELETE FROM planograma WHERE nro = OLD.nro AND num_serie = OLD.num_serie AND fabricante =
OLD.fabricante;
  RETURN OLD;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER remove_shelf_trigger
  BEFORE DELETE OR UPDATE
  ON prateleira
  FOR EACH ROW
  EXECUTE PROCEDURE remove_shelf();



-- Table: planograma (Trigger)
CREATE OR REPLACE FUNCTION remove_planogram()
RETURNS TRIGGER AS
$$
BEGIN
  DELETE FROM evento_reposicao WHERE nro = OLD.nro AND num_serie = OLD.num_serie AND
fabricante = OLD.fabricante;
  RETURN OLD;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER remove_planogram_trigger
  BEFORE DELETE OR UPDATE
  ON planograma
  FOR EACH ROW
  EXECUTE PROCEDURE remove_planogram();



-- Table: produto (Trigger)
CREATE OR REPLACE FUNCTION remove_product()
RETURNS TRIGGER AS
$$
BEGIN
  DELETE FROM planograma WHERE ean = OLD.ean;
  RETURN OLD;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER remove_product_trigger
  BEFORE DELETE OR UPDATE
  ON produto
  FOR EACH ROW
  EXECUTE PROCEDURE remove_product();
```

SQL

Schema: Triggers (continuação)

```
-- Table: categoria (Trigger)
CREATE OR REPLACE FUNCTION remove_category()
RETURNS TRIGGER AS
$$
DECLARE
  nome_categoria VARCHAR(100);
BEGIN
  nome_categoria := OLD.nome;

  DELETE FROM tem_outra WHERE categoria = nome_categoria;

  -- Checks if the category is simple or super and deletes it
  -- from its respective table.
  IF nome_categoria IN (SELECT nome FROM categoria_simples) THEN
    DELETE FROM categoria_simples WHERE nome = nome_categoria;

  ELSIF nome_categoria IN (SELECT nome FROM super_categoria) THEN
    DELETE FROM super_categoria WHERE nome = nome_categoria;

  END IF;

  DELETE FROM tem_categoria WHERE nome = nome_categoria;
  DELETE FROM produto WHERE cat = nome_categoria;
  DELETE FROM prateleira WHERE nome = nome_categoria;
  DELETE FROM responsavel_por WHERE nome_cat = nome_categoria;

  RETURN OLD;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER remove_category_trigger
  BEFORE DELETE OR UPDATE
  ON categoria
  FOR EACH ROW
  EXECUTE PROCEDURE remove_category();


-- Table: super_categoria (Trigger)
CREATE OR REPLACE FUNCTION remove_super_category()
RETURNS TRIGGER AS
$$
DECLARE
  num_sub INTEGER;
  nome_sub VARCHAR(100);
  nome_categoria VARCHAR(100);
BEGIN
  nome_categoria := OLD.nome;

  -- Stores in 'num_sub' the number of sub_categories of a
  -- super category.
  SELECT COUNT(categoria) INTO num_sub
  FROM tem_outra
    WHERE super_categoria = nome_categoria;

  -- Loops over all the sub categories and deletes them.
  WHILE num_sub != 0
  LOOP
    SELECT categoria INTO nome_sub
    FROM tem_outra
      WHERE super_categoria = nome_categoria
    LIMIT 1;

    DELETE FROM categoria WHERE nome = nome_sub;

    -- updates the 'num_sub' variable.
    SELECT COUNT(categoria) INTO num_sub
    FROM tem_outra
      WHERE super_categoria = nome_categoria;
  END LOOP;

  RETURN OLD;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER remove_super_category_trigger
  BEFORE DELETE OR UPDATE
  ON super_categoria
  FOR EACH ROW
  EXECUTE PROCEDURE remove_super_category();
```

```
-- Table: retalhista (Trigger)
CREATE OR REPLACE FUNCTION remove_retailer()
RETURNS TRIGGER AS
$$
DECLARE
  tin_retalhista INTEGER;
  num_cat_responsavel INTEGER;
  nome_categoria VARCHAR(100);
BEGIN
  tin_retalhista := OLD.tin;

  DELETE FROM evento_reposicao WHERE tin = tin_retalhista;

  -- Stores in 'num_cat_responsavel' the number of categories
  -- the retailer is responsible for.
  SELECT COUNT(nome_cat) INTO num_cat_responsavel
  FROM responsavel_por
    WHERE tin = tin_retalhista;

  -- Loops over all the categories and deletes them.
  WHILE num_cat_responsavel != 0
  LOOP
    SELECT nome_cat INTO nome_categoria
    FROM responsavel_por
      WHERE tin = tin_retalhista
    LIMIT 1;

    DELETE FROM categoria WHERE nome = nome_categoria;

    SELECT COUNT(nome_cat) INTO num_cat_responsavel
    FROM responsavel_por
      WHERE tin = tin_retalhista;
  END LOOP;

  RETURN OLD;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER remove_retailer_trigger
  BEFORE DELETE OR UPDATE
  ON retalhista
  FOR EACH ROW
  EXECUTE PROCEDURE remove_retailer();
```

SQL

View 'Vendas'

```
-------------------------------------------------------------------------------------------------------
-- vendas(ean , cat, ano , trimestre, dia_mes, mes , dia_semana , distrito, concelho, unidades)     --
--                                                                                                   --
--            dia_mes: [0, 31]                                                                       --
--            mes: [0, 12]                                                                           --
--            dia_semana: [0, 7]                                                                     --
-------------------------------------------------------------------------------------------------------

DROP VIEW IF EXISTS vendas;

CREATE VIEW vendas(ean, cat, ano, trimestre, dia_mes, mes, dia_semana, distrito, concelho, unidades) AS
  SELECT evento.ean,
      prod.cat,
      EXTRACT(YEAR from evento.instante) ano,
      EXTRACT(QUARTER from evento.instante) trimestre,
      EXTRACT(MONTH from evento.instante) mes,
      EXTRACT(DAY from evento.instante) dia_mes,
      EXTRACT(DOW from evento.instante) dia_semana,
      ponto_ret.distrito,
      ponto_ret.concelho,
      evento.unidades
    FROM evento_reposicao evento
    INNER JOIN produto prod
      ON evento.ean = prod.ean
    INNER JOIN instalada_em inst
      ON evento.num_serie = inst.num_serie AND evento.fabricante = inst.fabricante
    INNER JOIN ponto_de_retalho ponto_ret
      ON inst.loc = ponto_ret.nome

);
```

```
-- 1. Num dado periodo (i.e. entre duas datas), por dia da semana, por concelho e no total.
SELECT dia_semana, concelho, SUM(unidades) AS TotalSales
FROM vendas v
  WHERE make_date(ano::int, dia_mes::int, mes::int)
    BETWEEN '2020-01-01'::date AND '2024-02-23'::date
GROUP BY
  CUBE(dia_semana, concelho)
```

	dia_semana double precision	concelho character varying (100)	totalsales bigint
1	[null]	[null]	60
2	2	Loule	2
3	2	Amadora	13
4	5	Loule	1
5	4	Loule	3
6	6	Washington DC	5
7	4	Lagos	2
8	6	Lagos	4
9	1	Lagos	10
10	2	Vila nova de mil fontes	8
11	5	Vila nova de mil fontes	4
12	3	Lagos	8
13	3	[null]	8
14	6	[null]	9
15	2	[null]	23
16	4	[null]	5
17	5	[null]	5
18	1	[null]	10
19	[null]	Vila nova de mil fontes	12
20	[null]	Washington DC	5
21	[null]	Amadora	13
22	[null]	Lagos	24
23	[null]	Loule	6

Escolhemos para esta primeira consulta analisar as vendas no período entre **2020-01-01** e **2024-02-23.**
Podemos verificar que foram vendidos, no total, 60 produtos, sendo o dia com mais vendas **terça-feira (2)**, e o concelho com mais vendas, **Lagos**.
Os dias com menos vendas foram **quinta-feira (4)** e **sexta-feira (5)** ambos com 5 produtos vendidos, em relação ao concelho, **Washington DC** foi o com menos vendas.

Usando o **CUBE**, vamos gerar o grouping set:

- **((dia_semana, concelho), (dia_semana), (concelho), ())**

Este pode ser utilizado na query para obter as respostas às múltiplas interrogações simultaneamente. O valor **null** deve ser interpretado como "todos", ou seja, na entrada **(1, null, x)**, **x** corresponde ao número total de vendas realizadas na segunda-feira, somando as vendas em cada concelho que ocorreram neste mesmo dia.

SQL

View 'Vendas' (continuação)

```
-- 2. Num dado distrito (i.e.'Lisboa'), por concelho, categoria, dia da semana e no total.
SELECT concelho, cat, dia_semana, SUM(unidades) AS TotalSales
FROM vendas v
  WHERE distrito = 'Lisboa'
GROUP BY
  CUBE(concelho, cat, dia_semana)
```

	concelho character varying (100)	cat character varying (100)	dia_semana double precision	totalsales bigint
1	[null]	[null]	[null]	13
2	Amadora	Gomas	2	12
3	Amadora	Batatas	2	1
4	Amadora	Gomas	[null]	12
5	Amadora	Batatas	[null]	1
6	Amadora	[null]	[null]	13
7	[null]	Gomas	2	12
8	[null]	Batatas	2	1
9	[null]	Batatas	[null]	1
10	[null]	Gomas	[null]	12
11	Amadora	[null]	2	13
12	[null]	[null]	2	13

Escolhemos para esta consulta analisar as vendas no distrito de **Lisboa**.
Pela observação da tabela resultante, podemos concluir que foram vendidos um total de 13 produtos, todos eles vendidos no concelho da **Amadora**, a uma **terça-feira (2)**, sendo a categoria de produto mais vendida, **Gomas (12)**, e a menos vendida, **Batatas (1)**.

Idem da consulta OLAP anterior, usamos o **CUBE** para obter o grouping set:

- **((concelho, cat, dia_semana), (concelho), (cat), (dia_semana), ())**.

Tal como na análise anterior, o **null** deve ser interpretado como "todos", ou seja **(null, null, null)** indica o total de vendas em todos os concelhos, entre todas as categorias e dias da semana.

SQL

Índices

```
    ------------------------------------------------
    --                      7.1                   --
    ------------------------------------------------
    --  SELECT DISTINCT R.nome                    --
    --  FROM retalhista R, responsavel_por P      --
    --  WHERE R.tin = P.tin and P.nome_cat = 'Fruta' --
    ------------------------------------------------

CREATE INDEX index_tin_aux ON retalhista USING HASH(tin);
CREATE INDEX index_tin ON responsavel_por USING HASH(nome_cat);
```

Para a primeira query escolhemos utilizar **HASH** para ambas as tabelas **retalhista** e **responsavel_por**, nas colunas **tin** e **nome_cat**, respetivamente.
Desta forma tornamos mais eficientes as comparações **R.tin = P.tin** e **P.nome_cat = 'Fruta'** .

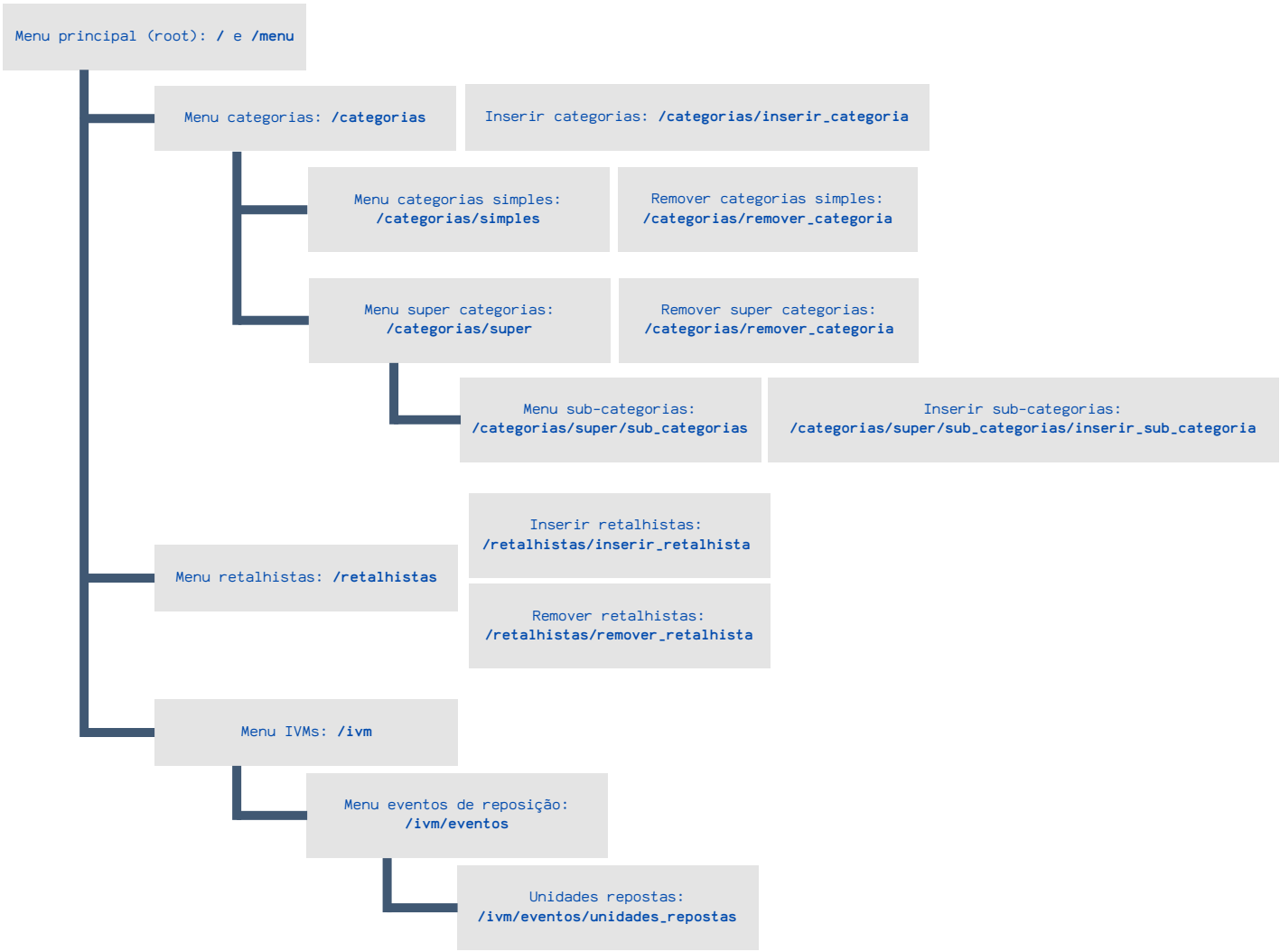
```
    ------------------------------------------------
    --                      7.2                   --
    ------------------------------------------------
    --  SELECT T.nome, count(T.ean)               --
    --  FROM produto P, tem_categoria T           --
    --  WHERE P.cat = T.nome and P.descr like 'A%'  --
    --  GROUP BY T.nome                           --
    ------------------------------------------------

CREATE INDEX index_nome ON produto USING HASH(cat);
CREATE INDEX index_descr ON produto USING BTREE(descr);
```

Para a segunda query, utilizamos **HASH** na tabela **produto**, na coluna **cat**, para assim, tornar mais eficiente a comparação **P.cat = T.nome**.
Utilizamos para além disso uma **BTREE** também na tabela **produto**, coluna **descr**, para evitar comparações desnecessárias, visto que a **BTREE** permite desprezar blocos de dados irrelevantes, tornando assim a procura **P.descr like 'A%'** muito mais eficiente.

SQL
Web app

A web app desenvolvida para acompanhar a base de dados foi feita utilizando a framework Flask e possui a seguinte estrutura:



Link para a app: http://web2.tecnico.ulisboa.pt/~ist199343/app.cgi/

Instruções de utilização:

Inserir categorias

- A partir do menu principal, entrar no menu categorias e selecionar 'Inserir categoria', e daí, especificar o nome e tipo da nova categoria e clicar em **Inserir**.

Remover categorias/sub-categorias

- A partir do menu principal, entrar no menu categorias e selecionar 'Categoria simples' ou 'Super categoria' consoante o tipo de categoria que se pretende remover, daí, escolher qual a categoria a remover e clicar em **Remover**.

Visualizar/Inserir sub-categorias

- A partir do menu principal, entrar no menu categorias, seguido do menu super categorias, e daí, clicar em 'Sub categorias', para as visualizar, depois disto, clicar em 'Inserir Sub-categorias' para inserir a sub-categoria que se pretende na super categoria selecionada (Por questão de conveniência a sub-categoria inserida é sempre uma simples).

Visualizar todos os eventos de reposição de uma IVM e unidades repostas por categoria de produto

- A partir do menu principal, entrar no menu IVMs e escolher qual a IVM que se pretende, para assim conseguir visualizar todos os seus eventos de reposição, de seguida clicar em 'Unidades repostas' para listar as unidades repostas por categoria de produto dessa mesma IVM.

Inserir/Remover retalhistas

- A partir do menu principal, entrar no menu retalhistas e clicar em 'Inserir retalhista' ou 'Remover retalhista', consoante o que se quer, inserir a informação necessária e confirmar.