UNIVERSIDADE FEDERAL DE UBERLÂNDIA

FACULDADE DE ENGENHARIA ELÉTRICA

TRABALHO FINAL

BANCO DE DADOS – CRIAÇÃO BASE DE DADOS

13/10/2020

FELIPE NARIMATSU PRESTI

VICTOR GABRIEL CASTÃO DA CRUZ

UBERLÂNDIA

2020

1 - INTRODUÇÃO

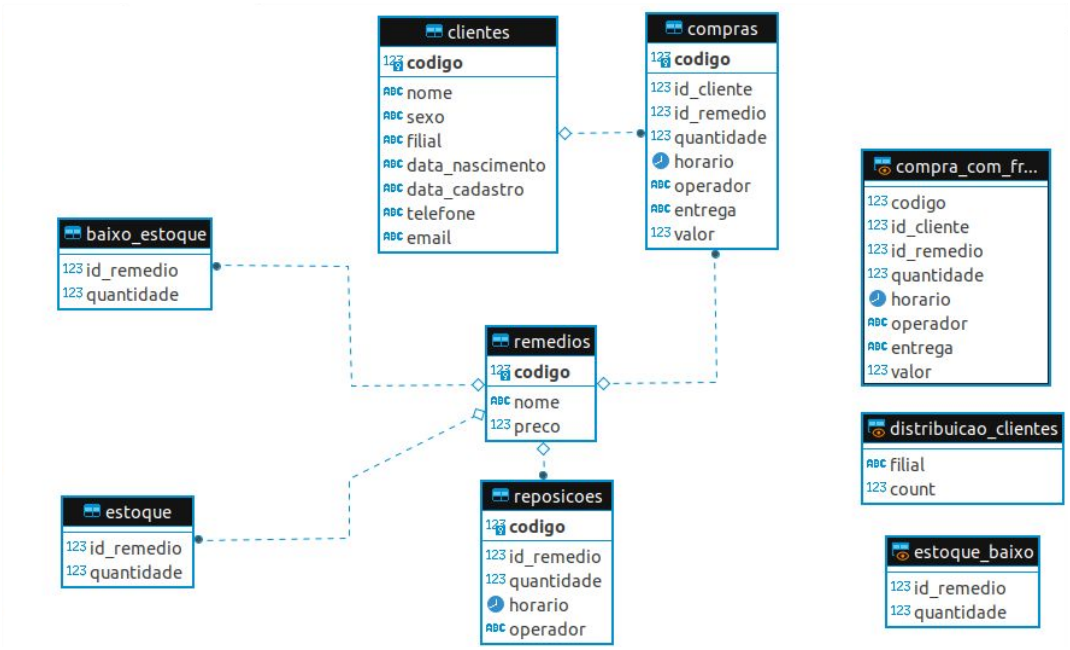
Com o objetivo de demonstrar uma das diversas aplicações de banco de dados, o projeto a ser desenvolvido consiste na criação de um relativo a uma farmácia.

Este banco, por sua vez, conterá tabelas responsáveis por armazenar dados a respeito de clientes e medicamentos, além de possuir funções que descrevem eventos que podem ocorrer neste determinado ambiente, como reposição de estoque, venda de medicamentos e até mesmo o cancelamento dessas vendas.

Desta forma, espera-se que, com o desenvolvimento do projeto, seja possível simular o funcionamento desse ambiente e exemplificar a grande utilidade que essas estruturas possuem atualmente.

2 - DESENVOLVIMENTO

Considerando as características da base de dados a ser desenvolvida, como views, tabelas e a relação que estas possuem entre si, seu modelo lógico, após as devidas construções, pode ser exemplificado a partir da seguinte figura:



2.1 - CRIAÇÃO DO SCHEMA

A partir da base de dados já construída no SGBD (nesse caso, o DBeaver), o schema "farmacia" deveria ser criado, pois a partir dele as tabelas, funções e todos os outros atributos do projeto seriam desenvolvidos.

A criação se deu pela seguinte linha de comando:

```
create schema farmacia;
```

2.2 – CRIAÇÃO DE TABELAS

O projeto desenvolvido necessitou da criação de seis tabelas, que serão descritas a seguir, juntamente com os comandos executados para tal.

2.2.1 - CLIENTES

A tabela seria composta por dados pessoais dos clientes da farmácia, cujas informações, para determinados campos, poderiam ou não estarem preenchidas. Além disso, o código de cada cliente seria utilizado como uma chave primária:

```
create table farmacia.clientes(
codigo integer PRIMARY KEY,
nome varchar(100),
sexo varchar(1),
filial varchar(10),
data_nascimento varchar(20),
data_cadastro varchar(20),
telefone varchar(15),
email varchar(40)
);
```

2.2.2 - REMÉDIOS

Tabela responsável por conter informações básicas do medicamento, sendo estas código (a ser usado como chave primária), nome e preço:

```
create table farmacia.remedios(
codigo integer primary key,
nome varchar(100),
preco numeric(7,2)
);
```

2.2.3 - ESTOQUE

Tabela contendo apenas o código do remédio e sua quantidade disponível. Nesse caso, o código seria uma chave estrangeira (referenciada pelo código da tabela do item 2.2.2):

```
create table farmacia.estoque(
id_remedio integer,
quantidade integer,
constraint fk_id_remedio foreign key (id_remedio) references farmacia.remedios(codigo)
);
```

2.2.4 - COMPRAS

A tabela em questão seria uma espécie de lista contendo informações a respeito dos remédios vendidos, como valor a pagar, código do remédio vendido e quantidade. Além disso, tanto o horário quanto o usuário em questão seriam adicionados, pois dessa forma, em um possível erro seria possível identificar quando a operação foi realizada e o responsável por ela.

A tabela de compras possui tanto chave primária quanto estrangeira, e foi criada a partir do seguinte comando:

```
create table farmacia.compras(
codigo integer primary key,
id_cliente integer,
id_remedio integer,
quantidade integer,
horario timestamp,
operador varchar(50),
entrega char(1),
valor numeric(10,2),
constraint fk_id_cliente foreign key (id_cliente) references farmacia.clientes(codigo),
constraint fk_id_remedio foreign key (id_remedio) references farmacia.remedios(codigo)
);
```

2.2.5 - REPOSIÇÕES

A tabela de reposições foi feita de forma análoga a tabela do item 2.2.5. A diferença se dá no resultado, tendo em vista que essa é responsável pela entrada de medicamentos no estoque, enquanto a anterior lida com a retirada de medicamentos do estoque.

Chaves primárias e estrangeiras também se fazem presente nessa tabela. Além disso, informações acerca do usuário e horário também se mantiveram, com a mesma motivação da tabela anterior:

```
create table farmacia.reposicoes(
codigo integer primary key,
```

id_remedio integer,

quantidade integer,

horario timestamp,

operador varchar(50),

constraint fk_id_remedio foreign key (id_remedio) references farmacia.remedios(codigo)

);

2.2.6 - BAIXO_ESTOQUE

Esta última tabela criada funciona como uma espécie de lembrete, contendo uma lista de produtos que devem ser repostos urgentemente, devido a sua baixa disponibilidade no estoque da farmácia:

create table farmacia.baixo_estoque(

id_remedio integer,

quantidade integer,

constraint fk_id_remedio foreign key (id_remedio) references farmacia.remedios(codigo)

);

2.3 - POPULANDO TABELAS

Somente três tabelas foram preenchidas durante a criação do banco, a de CLIENTES, REMÉDIOS e ESTOQUE, para realizar os INSERTs nas tabelas foram utilizados scripts em Python todos muito semelhantes usando a biblioteca psycopg2, que faz a conexão com o banco de dados e utiliza funções próprias para enviar linhas de código, após a leitura e manipulação dos dados do arquivo CSV, é criada a linha do INSERT e depois enviada com o *conn.commit()*.

O script usado para popular a tabela CLIENTES foi:

```
import csv
import psycopg2
conn = psycopg2.connect(database="postgres", user="postgres", password="banco", host="localhost",port=5432)
cur = conn.cursor()
with open('/home/felipenarimatsu/Documents/bd/cadastros-farmacia.csv', 'r') as csv_file:
    csv_reader = csv.reader(csv_file)
    for line in csv_reader:
        x=line
        x[0]=int(x[0])
        print(x)
        cur.execute("INSERT INTO farmacia.clientes VALUES (%s ,%s , %s, %s, %s, %s,%s ,%s)", x)
        conn.commit()
```

Para preencher a tabela dos medicamentos o código foi similar ao usado para preencher os clientes, porém este acessa somente o nome no arquivo CSV e gera o CODIGO do remédio no próprio python, através de um INT que é aumentado de um em um durante as iterações.

A última tabela preenchida com um script em Python, ESTOQUE, foi populada sem acessar um arquivo CSV, os valores foram criados direto no código, usando INTs para o código e quantidade de medicamentos.

2.4 - CRIAÇÃO DE FUNÇÕES

Em uma farmácia, sabe-se que compras, vendas, trocas e devoluções de remédios afetam diretamente algumas áreas, sendo a principal delas o estoque.

Dependendo do tamanho da farmácia, o gerenciamento dos estoques de forma manual pode gerar inconsistências e sérios problemas, tendo em vista o grande volume de operações que são realizadas a todo momento.

Com base nisso, quatro funções foram criadas para auxiliarem nesse processo, e serão descritas a seguir.

2.4.1 - FN_COMPRA_REMEDIO

Tal função é responsável pelo processo de compra de determinado medicamento por um cliente da farmácia, atuando também na quantidade de medicamentos disponíveis. Possui restrições impedindo que clientes inexistentes e produtos insuficientes (ou não cadastrados) sejam processados.

Além disso, a função apresenta o parâmetro “entrega”, que em caso positivo, adiciona R$ 10,00 ao custo total da compra (relativo ao custo do frete), permitindo uma simulação extremamente próxima ao que realmente ocorre em situações de compra de produtos:

```
create or replace function farmacia.fn_compra_remedio (id_compra integer, id_c
integer, id_r integer, qtde integer, entrega char) returns text as
$$
   declare
        linha record;
   begin
        select * into linha from farmacia.clientes where codigo = id_c;
        if not found then
             raise exception 'CLIENTE NÃO ENCONTRADO!';
        else
             select * into linha from farmacia.estoque where id_remedio = id_r;
             if not found then
```

```
                raise exception 'REMÉDIO NÃO ENCONTRADO!';
            elsif linha.quantidade < qtde then
                raise exception 'QUANTIDADE INSUFICIENTE!';
            else
                update farmacia.estoque
                set quantidade = quantidade - qtde where id_remedio = id_r;
                if entrega='N' then
                    insert into farmacia.compras values (id_compra, id_c, id_r, qtde, now(), user, entrega, qtde*(SELECT preco from farmacia.remedios WHERE codigo=id_r ));
                elseif entrega='S' then
                    insert into farmacia.compras values (id_compra, id_c, id_r, qtde, now(), user, entrega, (qtde*(SELECT preco from farmacia.remedios WHERE codigo=id_r ))+10);
                else
                    raise exception 'ENTREGA NÃO DEFINIDA!';
                end if;
                return 'COMPRA REALIZADA!';
            end if;
        end if;
    end
$$
language plpgsql;
```

2.4.2 - FN_REMOVE_COMPRA

É importante atentar-se ao fato de processos de compra podem, muitas vezes, conter erros que só são identificados após finalizar o processo. Desta forma, a função construída visa reparar procedimentos feitos de forma inadequada. Além disso, a função também atualiza o estoque (repondo os medicamentos que antes seriam retirados), e desta forma, impede previamente a inconsistência desses dados:

```
create or replace function farmacia.fn_remove_compra(id_compra integer) returns text as
$$
  declare
      linha record;
  begin
      select * into linha from farmacia.compras where codigo = id_compra;
      update farmacia.estoque set quantidade = quantidade + linha.quantidade where id_remedio = linha.id_remedio;
      delete from farmacia.compras where codigo=id_compra;
      return 'COMPRA CANCELADA!';
  end;
$$
language plpgsql;
```

2.4.3 - FN_REPOSICAO

Processa a reposição de determinado medicamento ao estoque, com a única restrição de que o código informado deve ser de um medicamento já cadastrado na tabela de remédios:

```
create or replace function farmacia.fn_reposicao (id_reposicao integer, id_r integer,
qtde integer) returns text as
$$
   declare
        linha record;
   begin
        select * into linha from farmacia.estoque where id_remedio = id_r;
        if not found then
             raise exception 'REMÉDIO NÃO ENCONTRADO!';
        else
             update farmacia.estoque
             set quantidade = quantidade + qtde where id_remedio = id_r;
              insert into farmacia.reposicoes values (id_reposicao, id_r, qtde,
now(), user);
             return 'REPOSIÇÃO REALIZADA!';
        end if;
   end
$$
language plpgsql;
```

2.4.4 - FN_VERIFICAR_ESTOQUE

A função em questão não age por ordem do usuário, mas sim devido ao disparo de um trigger. Tal disparo ocorre após o ato de inserir ou atualizar os dados da tabela de estoque.

Assim, sempre que a quantidade de um medicamento é alterada, a função identifica essa alteração e caso seja detectado um estoque considerado baixo (nesse caso, inferior a dez unidades), as informações são adicionadas à tabela de baixo estoque. Com isso, é possível realizar a reposição de um item antes de sua disponibilidade alcançar um nível crítico:

```
create or replace function farmacia.fn_verificar_estoque () returns trigger as
$$
  begin
      delete from farmacia.baixo_estoque where id_remedio = new.id_remedio;
      if (new.quantidade < 10) then
          insert into farmacia.baixo_estoque select new.id_remedio,
new.quantidade;
      end if;
      return new;
  end;
$$
language plpgsql;
```

Vale destacar que o ato de sempre deletar uma linha, e depois inseri-la novamente (caso a condição ainda seja verdadeira) foi uma medida tomada para evitar dados duplicados na tabela.

2.5 - CRIAÇÃO DE TRIGGERS

O trigger desenvolvido no projeto tem por objetivo impedir problemas relacionados a disponibilidade de medicamentos. Para isso, a cada atualização existente no estoque de medicamentos, dispara-se essa trigger, que percorre todas as linhas existentes na tabela de estoque.

Para cada linha alterada, uma função devidamente criada é executada e realiza todos os procedimentos necessários para evitar os problemas especificados. Essa função pode ser encontrada na descrição do item 2.4.4:

```
create trigger tg_verificar_estoque

after insert or update on farmacia.estoque

for each row execute procedure farmacia.fn_verificar_estoque();
```

Vale destacar que o trigger será majoritariamente disparado após atualizações na tabela. Entretanto, a condição de inserir também foi especificada considerando a possibilidade de um novo medicamento ser cadastrado já em condição de estoque baixo.

2.6 - CRIAÇÃO DE VIEWS

Views são extremamente importantes pelo fato de resumirem em apenas um nome uma série de comandos “select” que muitas vezes são complexos e, em caso de constante utilização, fariam de sua execução uma tarefa extremamente cansativa.

Para exemplificar as facilidades que as view apresentam, foram criadas três estruturas desse tipo, que serão descritas abaixo:

2.6.1 - ESTOQUE_BAIXO

Ao ser selecionada, informa quais medicamentos necessitam de uma rápida reposição no estoque.

```
create view farmacia.estoque_baixo as

select * from farmacia.baixo_estoque; --resultado da view
```

2.6.2 - COMPRA_COM_FRETE

Exibe quais das compras realizadas necessitarão de um serviço de entrega:

```
create view farmacia.compra_com_frete as

select * from farmacia.compras where entrega = 'S'; --resultado da view
```

2.6.3 - DISTRIBUIÇÃO_CLIENTES

Exibe a quantidade de clientes existentes em cada filial da farmácia

```
create view farmacia.distribuicao_clientes as

select filial, count(*) from farmacia.clientes group by filial;
```

2.7 - USUÁRIOS

Para criar usuários que atuam no banco, foram definidas três roles principais, o termo principais foi introduzido pois a definição de permissões é feita numa ROLE e os após a criação do usuário, é atribuído a ele sua respectiva role. Como último passo, os usuários são inseridos nos GROUPs, que existem para organizá-los e separá-los.

As ROLES criadas foram:

GERENTE: é o superusuário do banco, tem todas as permissões tanto do SCHEMA quanto das TABLE.

REPOSITOR: é o funcionário que tem o dever se arrumar o estoque físico e o estoque no banco de dados, por isso ele tem acesso ao uso do SCHEMA, SELECTs em todas as tabelas e UPDATEs somente em tabelas relacionadas com o estoque (*estoque, baixo_estoque e reposicoes*). Ele tem acesso a SELECTs em todas as tabelas para visualizar todo dado inserido, mas sem poder alterá-los.

VENDEDOR: é o funcionário que realiza as compras e as insere no banco de dados, por isso ele tem acesso ao uso do SCHEMA, SELECTs em todas as tabelas e UPDATEs na tabela de compras (*compras*). Ele tem acesso a SELECTs em todas as tabelas para poder verificar os estoque para poder realizar a compra sem maiores problemas.

O código implementado para criar as ROLES, USERS e GROUPS foi:

```
create role gerente with password 'gerente';
grant all on schema farmacia to gerente;
grant all ON table farmacia.clientes, farmacia.remedios, farmacia.estoque, farmacia.baixo_estoque
farmacia.reposicoes, farmacia.compras to gerente;
```

```
create role repositor with password 'repositor';
grant usage on schema farmacia to repositor;
grant select on table farmacia.clientes, farmacia.remedios, farmacia.estoque, farmacia.baixo_estoque,
farmacia.reposicoes, farmacia.compras TO repositor;
grant update on table farmacia.estoque, farmacica.baixo_estoque, farmacia.reposicoes to repositor;

create role vendedor with password 'vendedor';
grant usage on schema farmacia to vendedor;
grant select on table farmacia.clientes, farmacia.remedios, farmacia.estoque, farmacia.baixo_estoque
farmacia.reposicoes, farmacia.compras TO vendedor;
grant update on table farmacia.compras to vendedor;

create group grupo_gerente;
create group grupo_repositor;
create group grupo_vendedor;
```

Esse modo de implementar restrições permite que possa ser feita alterações mais facilmente e de modo mais direcionado, pois pode ser dado um GRANT ou REVOKE individual, sem alterar permissões de todo o grupo.

Implementações no mundo real são, funcionários que exercem duas ou mais funções na empresa, podendo ser inseridos em múltiplos grupos ou em casos de treinamento em outros setores, onde é garantido novas permissões mas sem efetuar a transição de grupo.

Os comandos para criação, garantir permissões e adicionar a grupos são:

```
create user USUARIO with password 'SENHA';
grant VENDEDOR to USUARIO;
alter group grupo_vendedor add user USARIO;
```

2.8 - OPERAÇÕES NO BANCO DE DADOS

Para que a aplicação do banco de dados pudesse ser compreendida, algumas operações e funções foram executadas, gerando como consequência alterações nas tabelas e ações tomadas de forma automática.

Inicialmente, simulou-se uma compra considerável de determinado medicamento através do seguinte comando:

select farmacia.fn_compra_remedio(1,30,30,40,'S');

Com isso, a quantidade disponível diminuiu significativamente, e com isso, o produto passou a ter menos de 10 unidades disponíveis.

Por esse motivo, disparou-se o trigger criado e, de forma automática, tal produto foi inserido na tabela de estoque baixo, indicando a necessidade de reposição.

Após a compra, executou-se a função de reposição a fim de verificar as alterações que seriam feitas:

select farmacia.fn_reposicao(1,30, 40);

Dessa forma, o produto foi reposto com sucesso e sua disponibilidade aumentou, o que fez com que tal medicamento fosse removido da tabela de estoque baixo.

Outra situação completamente comum é realizar incorretamente uma compra. Supondo que a compra realizada foi:

select farmacia.fn_compra_remedio(1,40,30,49,'S');

Entretanto, o código do cliente foi definido incorretamente. Para solucionar isso, basta chamar a seguinte função:

select farmacia.fn_remove_compra(1);

Após, basta executar a função de compra com os dados corretos e sem se preocupar com a compra antiga, pois a função acima se encarrega de resolver pendências relacionadas ao estoque dos medicamentos.

3 - CONCLUSÃO

O desenvolvimento de um banco de dados nem sempre é uma tarefa fácil, tendo em vista que, dependendo de sua área de atuação, os dados envolvidos podem ser complexos e estarem suscetíveis a erros.

O projeto desenvolvido pôde expressar o funcionamento de uma farmácia através de seu banco de dados, onde uma série de valores podem ser alterados à medida que operações como compra e venda ocorrem. Além disso, um dado pode influenciar em outro de forma direta ou indireta, e todas essas possibilidades devem ser analisadas no momento do desenvolvimento.

Vale destacar que o projeto desenvolvido poderia, ainda, ser aprimorado através de algumas adaptações ou criação de novas tabelas, funções, entre outros. Entretanto, sua forma atual ainda é capaz de exemplificar como essa estrutura poderia ser aplicada em determinado local (nesse caso, uma farmácia), demonstrando suas vantagens e comprovando a grande importância que bancos de dados apresentam no mundo contemporâneo.