COMPUTAÇÃO UFCG

Banco de Dados 1

Roteiro Básico sobre VIEWS, PROCEDURES e TRIGGERS.

Neste roteiro, iremos praticar conceitos básicos de VIEWS, PROCEDURES e TRIGGERS, usando o banco de dados COMPANY, já usado em outros roteiros.

Instruções para submissão

Siga as instruções abaixo para produzir e enviar seu material:

- Crie um arquivo de texto chamado **roteiroXX-matricula.sql**
- Adicione os comandos utilizados ao arquivo, na **mesma ordem** que os executar.
- Use comentários no seu código para sinalizar onde cada questão do roteiro está começando e para anotar qualquer outra coisa que desejar. As linhas que denotam comentários iniciam com '--'.
- Quando concluir, envie o arquivo craiado utilizando o formulário de submissão de roteiros.

Materiais Úteis

Os seguintes links poderão ser úteis durante a elaboração do roteiro:

VIEWS:
https://www.postgresql.org/docs/9.5/static/sql-createview.html

FUNCTIONS / PROCEDURES (são conceitos equivalentes no PostgreSQL):
https://www.postgresql.org/docs/9.5/sql-createfunction.html

TRIGGERS:
https://www.postgresql.org/docs/9.5/sql-createtrigger.html

Antes de começar

Execute os arquivos de criação das tabelas e povoamento, **company.sql** e **company-data.sql** respectivamente.

Questões Propostas

1. Crie as seguintes VIEWS:

   A. **vw_dptmgr**: contém apenas o número do departamento e o nome do gerente (primeiro nome);
   B. **vw_empl_houston:** contém o ssn e o primeiro nome dos empregados com endereço em Houston;
   C. **vw_deptstats**: contém o número do departamento, o nome do departamento e o número de funcionários que trabalham no departamento;
   D. **vw_projstats:** contém o id do projeto e a quantidade de funcionários que trabalham no projeto;

2. Faça consultas nas VIEWS para checar se ficaram corretas;

3. Remova todas as VIEWS criadas;

4. Crie uma FUNCTION chamada *check_age*, que recebe como parâmetro o ssn de um empregado e retorna:
   - 'SENIOR', se o funcionário tem 50 anos ou mais;
   - 'YOUNG', se o funcionário tiver menos de 50 anos;
   - 'UNKNOWN', se a data de nascimento for NULL;
   - 'INVALID', se a data de nascimento for uma data futura;

Para facilitar, abaixo segue um template para criação da sua função:

```
CREATE OR REPLACE FUNCTION xxxxx(param_name PARAM_TYPE)
RETURNS RETURN_TYPE AS
$$

DECLARE
var1_name VAR1_TYPE;
var2_name VAR2_TYPE;
var3_name VAR3_TYPE;

BEGIN

       IF (CONDITION) THEN
              ...
```

```
        ELSIF (CONDITION) THEN
                ...
        END IF;

END;
$$  LANGUAGE plpgsql;
```

Em seguida, execute os comandos abaixo e confira os resultados.
Os resultados abaixo podem mudar com o passar do tempo, pois depende da data atual. Se notar alguma diferença, por favor me avise.

```
your_db=> SELECT check_age('666666609');
 check_age
-----------
 SENIOR
(1 row)

your_db=> SELECT check_age('555555500');
 check_age
-----------
 YOUNG
(1 row)

your_db=> SELECT check_age('987987987');
 check_age
-----------
 INVALID
(1 row)

your_db=> SELECT check_age('x');
 check_age
-----------
 UNKNOWN
(1 row)

your_db=> SELECT check_age(null);
 check_age
-----------
 UNKNOWN
(1 row)
```

Observe que sua função pode ser usada em suas consultas.

```
your_db=> SELECT ssn FROM employee WHERE check_age(ssn) = 'SENIOR';
    ssn
-----------
 111111100
 444444400
 111111101
 111111102
 111111103
 222222200
 222222201
 222222202
```

```
 222222203
 222222204
 222222205
 333333300
 333333301
 444444401
 444444402
 555555501
 666666600
 666666601
 666666602
 666666603
 666666604
 666666605
 666666606
 666666607
 666666608
 666666609
 666666610
 666666612
(28 rows)
```

5. É muito comum combinar FUNCTIONs e TRIGGERs para se definir CONSTRAINTs mais sofisticadas. O PostgreSQL inclui o conceito de TRIGGER PROCEDURES, que permite fazer esta combinação. Veja a documentação aqui e responda o que se pede.

Crie uma TRIGGER PROCEDURE chamada **check_mgr** que monitora as atualizações (INSERT/UPDATE) realizadas na **tabela *departament***, não permitindo que um gerente inadequado seja alocado a um departamento. Um gerente é considerado **inadequado se**:

1. não for um funcionário atualmente alocado no departamento; ou
2. não possuir subordinados; ou
3. não for 'SENIOR';

Lance as exceções conforme necessário:

```
RAISE EXCEPTION 'manager must be a department''s employee';

RAISE EXCEPTION 'manager must be a SENIOR employee';

RAISE EXCEPTION 'manager must have supevisees';
```

Sugestão para começar a testar sua trigger:

A. Caso já tenha criado, remova a trigger:

```
DROP TRIGGER check_mgr ON department;
```

B. Insira um departamento com dnumber = 2, tendo como gerente o empregado com ssn = '999999999'.

```
-- isso deve funcionar pois não existe uma FK para validar se o employee
existe
INSERT INTO department VALUES ('Test', 2, '999999999', now());
```

C. Insira um novo employee com ssn = '999999999' e que atenda aos requisitos 2 e 3 para ser gerente. Insira também um subordinado.

```
-- employee '999999999'
INSERT INTO employee VALUES
('Joao','A','Silva','999999999','10-OCT-1950','123 Peachtree, Atlanta,
GA','M',85000,null,2);

-- employee '999999998', subordinado ao anterior
INSERT INTO employee VALUES
('Jose','A','Santos','999999998','10-OCT-1950','123 Peachtree, Atlanta,
GA','M',85000,'999999999',2);
```

D. Crie a trigger;

E. Agora execute os comandos abaixo e verifique se obteve os mesmos resultados.

```
-- o update deve funcionar normalmente
your_db=> UPDATE department SET mgrssn = '999999999' WHERE dnumber=2;

-- não permite executar update
-- dependendo da forma como a trigger foi implementada, você pode ter uma
validação pela FK not-null ou pela trigger após executar o UPDATE:

your_db=> UPDATE department SET mgrssn = null WHERE dnumber=2;

ERROR:  null value in column "mgrssn" violates not-null constraint
```

```
DETAIL: … (se tiver sido validado pela FK)

-- OU
ERROR:  manager must be a department's employee (se tiver sido validado
pela trigger)


-- não permite executar update porque esse employee não existe
-- isso poderia ser verificado por uma FK department.mgrssn → employee.ssn,
mas essa FK não existe
your_db=> UPDATE department SET mgrssn = '999' WHERE dnumber=2;
ERROR:  manager must be a department's employee

-- não permite executar update pois o employee não é do departamento
your_db=> UPDATE department SET mgrssn = '111111100' WHERE dnumber=2;
ERROR:  manager must be a department's employee

-- altera a data de nascimento do employee para que ele deixe de ser Sênior
your_db=> UPDATE employee SET bdate = '10-OCT-2000' WHERE ssn =
'999999999';
UPDATE 1

-- não permite executar update
your_db=> UPDATE department SET mgrssn = '999999999' WHERE dnumber=2;
ERROR:  manager must be a SENIOR employee

-- altera a data de nascimento do employee para que ele volte a ser Sênior
UPDATE employee SET bdate = '10-OCT-1950' WHERE ssn = '999999999';
UPDATE 1

-- o update deve funcionar normalmente
your_db=> UPDATE department SET mgrssn = '999999999' WHERE dnumber=2;
UPDATE 1

--remove os subordinados
DELETE FROM employee WHERE superssn = '999999999';
DELETE 1

-- não permite executar update pois o empregado não tem subordinados
your_db=> UPDATE department SET mgrssn = '999999999' WHERE dnumber=2;
ERROR:  manager must have supevisees

--remove o employee '999999999'
DELETE FROM employee WHERE ssn = '999999999';
DELETE 1

--Remove o departamento 2
your_db=> DELETE FROM department where dnumber=2;
DELETE 1
```

Discussão:

Os comandos acima representam apenas alguns testes básicos, mas observe que eles não testam casos de erro durante a inserção de departamentos, por exemplo. Você pode fazer vários outros testes (não precisa enviar).

Os comandos acima apenas verificam se a constraint funciona bem para UPDATE. Se quiser, faça testes adicionais com INSERT também (não precisa enviar).

Observe que esta *constraint* ainda não está muito alinhada com o que provavelmente seriam as regras de negócio. Apenas alguns exemplos:

- O gerente do departamento 2 pode ser funcionário do mesmo departamento, mas seus subordinados serem funcionários de outros departamentos.
- Se os subordinados forem excluídos da tabela *employee* depois dos UPDATES terem sido executados na tabela *departament*, o banco passará a um estado inconsistente.
- Etc...

Se quiser brincar mais um pouco, tente aprimorar esta regra de negócio e atualizar a Trigger. **Não** é preciso enviar esta parte complementar, caso implemente.