LISTA 11 - RESPOSTAS

TRIGGERS

1 - Elabore uma função que será chamada por uma trigger, com o objetivo de impedir as inserções ou atualizações inválidas dos salários dos funcionários. A verificação deverá ser feita a partir do "job.id" e, caso o novo salário não estiver entre o valor mínimo e máximo do cargo, a mensagem "Salário INVÁLIDO!" deverá ser exibida. Teste a funcionalidade da trigger inserindo valores válidos e inválidos na tabela "employee", observando a tabela "job", em função do cargo e salário.

```
--DROP FUNCTION f_checksalary () CASCADE;


CREATE OR REPLACE FUNCTION f_checksalary ()
    RETURNS TRIGGER
    AS $$
    DECLARE
    minsal job.minsalary%TYPE;
    maxsal job.maxsalary%TYPE;
    BEGIN
        SELECT minsalary, maxsalary INTO minsal, maxsal
        FROM job
        WHERE id = NEW.job_id;
        IF NEW.salary NOT BETWEEN minsal AND maxsal THEN
            RAISE NOTICE 'Salário INVÁLIDO!';
            RETURN NULL;
        ELSE
            RETURN NEW;
        END IF;
    END;
    $$ LANGUAGE plpgsql;


CREATE TRIGGER t_checksalary
BEFORE INSERT OR UPDATE ON employee
FOR EACH ROW
EXECUTE PROCEDURE f_checksalary ();


INSERT INTO employee
(firstname, lastname, email, phone, hire, job_id, salary, commission,
manager_id, department_id)
VALUES
('Juliano','Dertzbacher','juliano.dertzbacher@univates.br',5190909090,'2015-10-
01',7,50000.00,NULL,NULL,40);
```

2 - Elabore uma função que será chamada por uma trigger, com o objetivo de impedir a atualização do ID na tabela "country" e, caso houver alguma tentativa, a mensagem "não é permitido atualizar o campo "country.id"!" deverá ser exibida. Teste a funcionalidade da trigger atualizando os valores do campo "id" na tabela "country".

```
--DROP FUNCTION f_updatecountry () CASCADE;


CREATE OR REPLACE FUNCTION f_updatecountry ()
    RETURNS TRIGGER
    AS $$
    BEGIN
        IF (NEW.id <> OLD.id) THEN
            RAISE NOTICE 'Não é permitido atualizar o campo "country.id"!';
            RETURN NULL;
        ELSE
          RETURN NEW;
        END IF;
    END;
    $$ LANGUAGE plpgsql;


CREATE TRIGGER t_updatecountry
BEFORE UPDATE ON country
FOR EACH ROW
EXECUTE PROCEDURE f_updatecountry ();


UPDATE country
SET id = 1000
WHERE id = 1;
```

3 - Elabore uma função que será chamada por uma trigger, com o objetivo de impedir a inserção de um novo presidente na tabela "employee" e, caso houver alguma tentativa, a mensagem "A empresa só pode ter um presidente!" deverá ser exibida.

Teste a funcionalidade da trigger inserindo um novo funcionário com o "job.id" igual a 19 e outros códigos válidos.

```
--DROP FUNCTION f_checkpresident () CASCADE;


CREATE OR REPLACE FUNCTION f_checkpresident ()
    RETURNS TRIGGER
    AS $$
    BEGIN
        IF (NEW.job_id = 19) THEN
            RAISE NOTICE 'A empresa só pode ter um presidente!';
            RETURN NULL;
        ELSE
          RETURN NEW;
        END IF;
    END;
    $$ LANGUAGE plpgsql;


CREATE TRIGGER t_checkpresident
BEFORE INSERT ON employee
FOR EACH ROW
EXECUTE PROCEDURE f_checkpresident ();


INSERT INTO employee
(firstname, lastname, email, phone, hire, job_id, salary, commission,
manager_id, department_id)
VALUES
('Juliano','Dertzbacher','juliano.dertzbacher2@univates.br',5190909090,'2015-
10-01',19,10000.00,NULL,NULL,40);
```

4 - Crie uma tabela chamada de "employee_delete" com a mesma estrutura da tabela "employee", utilizando o comando: "CREATE TABLE employee_delete (LIKE employee);". Elabore uma função que será chamada por uma trigger, com o objetivo de armazenar na tabela "employee_delete" todos os registros excluídos da tabela "employee".

Teste a funcionalidade da trigger excluindo um funcionário da tabela "employee" e verifique se o mesmo foi automaticamente incluído na tabela "employee_delete".

```
CREATE TABLE employee_delete (LIKE employee);


--DROP FUNCTION f_employeedelete () CASCADE;


CREATE OR REPLACE FUNCTION f_employeedelete ()
    RETURNS TRIGGER
    AS $$
    BEGIN
        INSERT INTO employee_delete
            SELECT *
            FROM employee
            WHERE id = OLD.id;
        RETURN OLD;
    END;
    $$ LANGUAGE plpgsql;


CREATE TRIGGER t_employeedelete
BEFORE DELETE ON employee
FOR EACH ROW
EXECUTE PROCEDURE f_employeedelete ();


DELETE FROM employee
WHERE id = 10;
```