Universidade de São Paulo
Escola de Artes, Ciências e Humanidades
Disciplina: Laboratório de Banco de Dados
Profª Dra. Fátima Nunes.

Administração de Condomínio

Parte II

Fernando K. G. de Amorim – 10387644
João Guilherme da Costa Seike – 9784634
Lucas Pereira Castelo Branco – 10258772
Victor Gomes de O. M. Nicola – 9844881

São Paulo
2019

LETRA B

Regras de negócios para a parte (b):

1. Ao associar um edifício novo ao condomínio, todos os apartamentos devem ser cadastrados de maneira automática e ordenada de acordo com o número do andar e com o número final de cada apartamento.

Enunciado textual

Para evitar esforço desnecessário durante a inserção de novos edifícios, onde seria necessário também realizar a inserção de todos os apartamentos deste edifício, foi implementado um gatilho que ao receber uma inserção de um edifício, automaticamente insere todos os apartamentos deste edifício na tabela "Apartamento".

Solução textual em SQL padrão

```
CREATE OR REPLACE TRIGGER tr_insere_apartamentos
AFTER INSERT OF adm_condominio.Edificio
FOR EACH ROW
DECLARE
  id_max integer;
BEGIN
  SELECT max(id_moradia) into id_m from adm_condominio.Moradia;
  FOR i in 1 .. new.andares
  LOOP
    FOR j in 1 .. new.qtd_finais
    LOOP
      id_m := id_m + 1;
      INSERT INTO adm_condominio.Moradia (tipo_moradia, id_moradia) VALUES
        ('a', (id_m));
      INSERT INTO adm_condominio.Moradia_Edificio (fk_id_edificio, fk_id_moradia) VALUES
        (new.id_edificio, (id_m));
      INSERT INTO adm_condominio.Condominio_Moradia (fk_id_condominio, fk_id_moradia) VALUES
        (new.fk_id_condominio, (id_m));
      INSERT INTO adm_condominio.Apartamento (fk_id_moradia,andar, final, numero_ap) VALUES
        (id_m, i, j, CAST(CONCAT(CAST(i as varchar(10)), CAST(j as varchar(10))) AS INT));
    END LOOP;
  END LOOP;
END;
```

Solução em código implementada

Esta é uma imagem da solução em código implementada dentro do SGBD PostgreSQL:

```
CREATE OR REPLACE FUNCTION fc_insere_apartamentos()
RETURNS trigger AS $tr_apartamentos$
DECLARE
    id_max integer;
BEGIN
    SELECT max(id_moradia) into id_m from adm_condominio.Moradia;

    FOR i in 1 .. new.andares
    LOOP
        FOR j in 1 .. new.qtd_finais
        LOOP

            id_m := id_m + 1;

            INSERT INTO adm_condominio.Moradia_Edificio (fk_id_edificio, fk_id_moradia) VALUES
                (new.id_edificio, (id_m));
            INSERT INTO adm_condominio.Condominio_Moradia (fk_id_condominio, fk_id_moradia) VALUES
                (new.fk_id_condominio, (id_m));

            INSERT INTO adm_condominio.Moradia (tipo_moradia, id_moradia) VALUES
                ('a', (id_m));

            INSERT INTO adm_condominio.Apartamento (fk_id_moradia,andar, final, numero_ap) VALUES
                (id_m, i, j, CAST(CONCAT(CAST(i as varchar(10)), CAST(j as varchar(10))) AS INT));
        END LOOP;
    END LOOP;

RETURN NEW;
END;
$teste_trigger$ LANGUAGE plpgsql;

CREATE TRIGGER tr_insere_apartamentos
AFTER INSERT ON adm_condominio.Edificio
FOR EACH ROW
EXECUTE PROCEDURE tr_insere_apartamentos();
```

Regras de negócios para a parte (b):

O código implementado funciona da seguinte maneira:
1- É criado a função fc_insere_apartamentos que será executada pelo trigger.
2- A função armazena o maior id_moradia da tabela adm_condominio.Moradia na variável id_m
3-a função entrea em um loop que vai iterar de 1 até a quantidade de andares do edifício inserido e a cada iteração do andar, é feita uma iteração de 1 até o numero de qtd_finais.
4- Dentro do loop de qtd_finais, o valor de id_m é incrementado em seguida é inserido o apartamento no banco na seguinte ordem: insere na tabela Moradia, insere na tabela Edificio_Moradia, insere Condominio_Moradia (sempre passando o valor do id incrementado) e finalmente insere em Apartamento o id e o numero do apartamento, que é a concatenação dos valores (i,j) da iteração atual (andar atual + numero do apartamento atual) que resulta no número completo do apartamento.
5-Por fim, é criado um trigger chamado tr_insere que será ativado quando toda vez que acontecer um Insert na tabela adm_condominio.Edificio e executará a função descrita acima(APÓS a inserção).

Regras de negócios para a parte (b):

2. Ao deletar um edifício da tabela Edificío, além de deletar todos os apartamentos vinculados a esse edifício na tabela Apartamento, também devem ser deletados todas as pessoas que estão relacionadas com esses apartamentos da tabela Pessoas.

Enunciado textual

Caso um edifício deixe de ser propriedade do condomínio (em caso de venda ou demolição do edifício, por exemplo), deve haver uma deleção na tabela Edifício que ativará um gatilho que automaticamente deleta todos os apartamentos deste edifício e todas as pessoas que moravam nesse edifício do banco de dados do condomínio.

Solução textual em SQL padrão

```
CREATE TRIGGER tr_deleta_apartamentos
AFTER DELETE OF adm_condominio.Edificio
FOR EACH ROW
BEGIN
    CREATE TABLE table_holder AS (SELECT fk_id_moradia FROM adm_condominio.Moradia_Edificio WHERE fk_id_edificio = old.id_edificio);
   DELETE FROM adm_condominio.Moradia_Pessoa WHERE fk_id_moradia IN (SELECT fk_id_moradia FROM table_holder);
      DELETE FROM adm_condominio.Condominio_Moradia WHERE  fk_id_moradia IN (SELECT fk_id_moradia FROM table_holder);
    DELETE FROM adm_condominio.Apartamento WHERE  fk_id_moradia IN (SELECT fk_id_moradia FROM table_holder);
  DELETE FROM adm_condominio.Moradia_Edificio WHERE fk_id_edificio = old.id_edificio;
    DELETE FROM adm_condominio.Moradia WHERE  id_moradia IN (SELECT fk_id_moradia FROM table_holder);
  DROP TABLE table_holder;
END;
```

Solução em código implementada

Esta é uma imagem da solução em código implementada dentro do SGBD PostgreSQL:

```sql
CREATE OR REPLACE FUNCTION fc_deleta_apartamentos()
RETURNS trigger AS $tr_dapartamentos$
BEGIN

  CREATE TABLE table_holder AS (SELECT fk_id_moradia FROM adm_condominio.Moradia_Edificio WHERE fk_id_edificio = old.id_edificio);


  DELETE FROM adm_condominio.Moradia_Pessoa WHERE fk_id_moradia IN (SELECT fk_id_moradia FROM table_holder);

  DELETE FROM adm_condominio.Condominio_Moradia WHERE  fk_id_moradia IN (SELECT fk_id_moradia FROM table_holder);
  DELETE FROM adm_condominio.Apartamento WHERE  fk_id_moradia IN (SELECT fk_id_moradia FROM table_holder);
  DELETE FROM adm_condominio.Moradia_Edificio WHERE fk_id_edificio = old.id_edificio;
  DELETE FROM adm_condominio.Moradia WHERE  id_moradia IN (SELECT fk_id_moradia FROM table_holder);
  DROP TABLE table_holder;
RETURN NEW;
END;
$tr_dapartamentos$ LANGUAGE plpgsql;

CREATE TRIGGER tr_deleta_apartamentos
before DELETE ON adm_condominio.Edificio
FOR EACH ROW
EXECUTE PROCEDURE fc_deleta_apartamentos();
```

Regras de negócios para a parte (b):

O código implementado funciona da seguinte maneira:

1- É criado a função fc_deleta_apartamentos que será executada pelo trigger.
2-Essa função cria uma tabela temporária e coloca todos os “id_moradia” da tabela Moradia_Edificio pois serão essas moradias que serão deletadas.
3-A tabela temporária é usada para deletar as informações referentes ao edifício deletado em outras tabelas, decidimos usar uma tabela temporária para armazenar os IDs das moradias deletados pois não era possível guardar todas essas informações em uma só variável.
4- Então é feito a deleção nas outras tabelas na seguinte ordem: Condominio_Moradia, Apartamento, Moradia_Edificio e Moradia.
5- Por fim é criado um gatilho que será ativado após acontecer um DELETE na tabela adm_condominio.Edificio e executará a função descrita acima(ANTES da deleção).