Universidade Federal Fluminense – UFF
Instituto de Computação
Departamento de Ciência da Computação

Restrições semânticas, *Triggers* e Funções em PostgreSQL

Projeto para Banco de Dados II

Alunos: João Matheus Arrruda Tavares
Patrick Pissurno
Rafael Duarte Campbell de Medeiros

Professor: Luis André Paes Leme

Niterói – RJ
Julho / 2019

Sumário

1 Introdução

1.1 Apresentação do modelo

O sistema pensado para o trabalho é baseado em uma empresa de consultoria que tem funcionários e os organiza em equipes, de forma que um mesmo funcionário pode estar em mais de uma equipe. Cada equipe é composta por um número ilimitado de funcionários e tem apenas um líder. Essa equipe assume projetos, de modo que cada projeto tenha apenas uma equipe. Esses projetos, por sua vez, fazem parte de uma e apenas uma categoria.

Uma parte importante do funcionamento da empresa é o sistema de permissão: ele determina quais projetos uma equipe poderá assumir. Cada funcionário terá um nível de permissão, que determina seu grau de relevância na empresa; o grau de permissão de uma equipe será, sempre, o grau de permissão do membro de menor nível. As categorias de projeto têm uma permissão necessária associada, que determina o grau de relevância deste projeto; deste modo, a permissão associada a um projeto é a permissão de sua categoria.

Uma possível modelagem **entidade-relacionamento** para o sistema está descrita no Diagrama 1.

1.2 Diagrama Entidade-Relacionamento

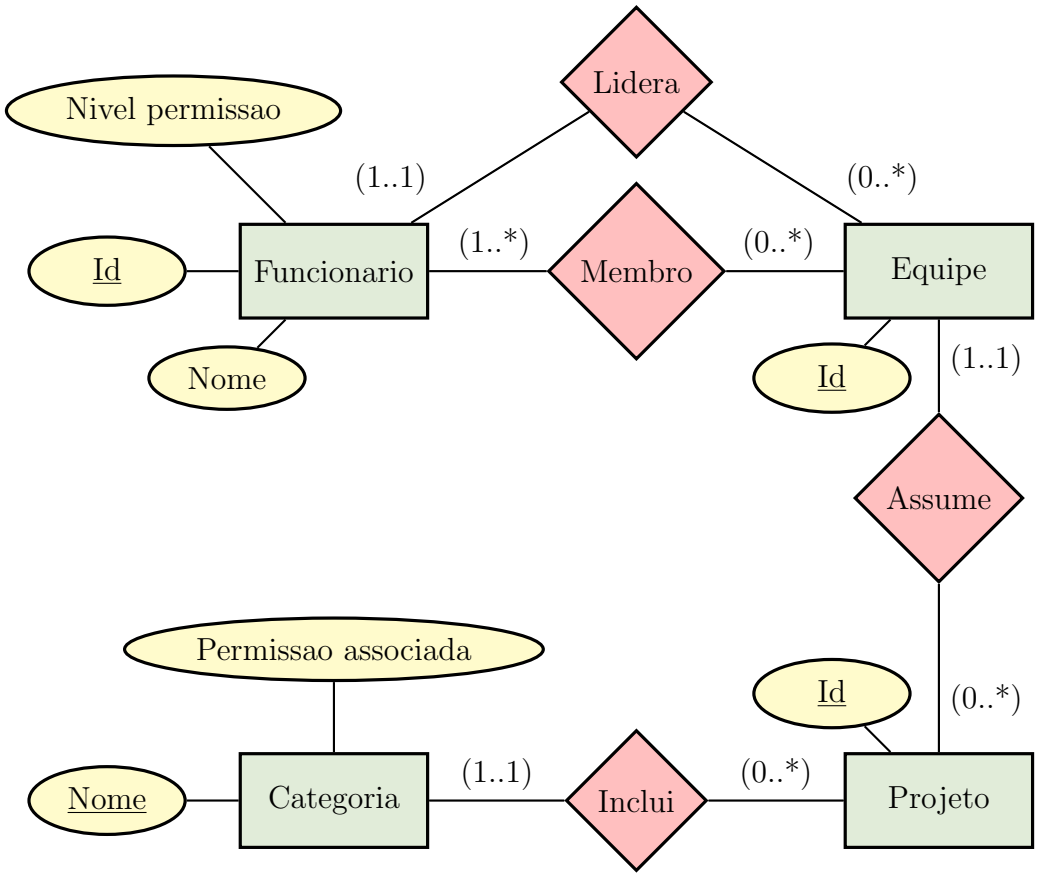

Diagrama 1: Entidade-relacionamento do projeto de banco de dados

2 Restrições semânticas

A primeira parte do trabalho consistia em sugerir três restrições de integridade semânticas, ou seja, três **regras de negócio** que devem ser respeitadas para garantir a consistência dos dados. Para cada restrição, era necessário identificar quais tabelas poderiam pô-la em risco e tratar, por meio de *triggers*, as atualizações, inserções e deleções.

Cada restrição será apresentada e, em seguida, é dado uma relação de quais tabelas devem ser verificadas, para quais operações e em qual momento. Ao final do documento, encontra-se a relação de todos os códigos utilizados para implementar e garantir as restrições.

Vale a pena ressaltar que uma restrição de integridade pode ser garantida de vários modos diferentes e, neste trabalho, tentamos seguir por dois caminhos diferentes:

- **Primeira restrição**: cada *trigger* está associado a uma tabela e, para ela, verificará se as modificações que se pretende fazer manterão a consistência dos dados. Desse modo, a verificação é feita **antes** da alteração.
- **Segunda e terceira restrição**: com o auxilio de uma função que verifica a consistência de uma regra, cada *trigger* só deve verificar se o banco ainda está consistente, **antes** ou **depois** das modificações serem feitas.

Para a criação do banco, optou-se pelo uso de *restrict* em quase todas as remoções de chave estrangeira para garantir que os elementos fossem isolados antes de serem removidos, simplificando as verificações necessárias nos *triggers*. O único caso onde foi utilizado *cascade* é na tabela equipes_funcionarios, onde a remoção só afetará a terceira restrição, cujo *trigger* está configurado para agir.

2.1 Restrição 1

A primeira restrição de integridade diz respeito, em especial, às equipes e seus projetos: **nenhum funcionário pode ter permissão inferior às permissões associada aos projetos da equipe que participa**.

Tabela	Operação	Momento
Categoria	Atualização	Antes
Projeto	Atualização e inserção	Antes
Funcionario	Atualização	Antes
Equipe	Atualização	Antes
Equipes_funcionarios	Atualização e inserção	Antes

2.2 Restrição 2

A segunda restrição pensada diz respeito, em especial, às equipes e seus líderes: **um líder deve ter permissão superior a todos os membros das equipes que lidera**.

Tabela	Operação	Momento
Funcionario	Atualização	Antes
Equipe	Atualização	Depois
Equipes_funcionarios	Atualização e inserção	Antes

2.3 Restrição 3

A terceira restrição diz respeito especialmente às equipes e seus projetos: **A soma das permissões das categorias dos projetos de uma equipe não deve ser maior do que a metade da soma de todas as permissões de seus membros**.

Tabela	Operação	Momento
Categoria	Atualização	Depois
Projeto	Atualização e inserção	Depois
Funcionario	Atualização	Depois
Equipes_funcionarios	Atualização ou deleção	Depois

3 Funções

Para as funções, o objetivo era criar rotinas que pudessem ser chamadas para realizar ações pontuais no banco de dados. Procurou-se construir funções que fizessem sentido no ambiente empresarial, considerando a dinâmica entre equipes e projetos.

Cada função pôde ser definida isoladamente, sem a necessidade de chamar outra função auxiliar. Ambas têm como retorno uma tabela, portanto, devem ser chamadas sempre por uma seleção.

3.1 Função 1

O objetivo da primeira função (Função 13) é, em suma, **retornar uma lista de sugestões para a criação de uma equipe**. Ela recebe como parâmetro uma **permissão mínima** desejada e, também, a ocupação máxima dos candidatos, ou seja, o **número máximo de projetos** que os candidatos podem estar participando.

Exemplo de *query*:

```
SELECT * FROM sugestoes_nova_equipe_projeto(3, 3);
```

Resultado da *query*:

Id	Nome	Nº projetos	Nível permissão
4	Andreia	1	6
2	Rosislene	2	3
1	Rodolfo	2	4

3.2 Função 2

O objetivo da segunda função (Função 14) é retornar uma relação dos funcionários e suas equipes; deste modo, deve retornar **uma lista completa de todas as equipes e sua relação com cada um dos funcionários, indicando se este é líder, membro ou não participa**. A função não recebe nenhum parâmetro.

Exemplo de *query*:

```
SELECT * FROM relatorio_funcionario_equipe();
```

Resultado da *query*:

Equipe	Funcionário	Cargo
1	Rodolfo (1)	Líder
1	Rosislene (2)	Membro
1	Mario (3)	Membro
1	Andreia (4)	Não participa
1	Leonardo (5)	Não participa
2	Rodolfo (1)	Líder
2	Rosislene (2)	Membro
2	Mario (3)	Não participa
2	Andreia (4)	Não participa
2	Leonardo (5)	Não participa
3	Rodolfo (1)	Não participa
3	Rosislene (2)	Não participa
3	Mario (3)	Não participa
3	Andreia (4)	Líder
3	Leonardo (5)	Membro

4 Códigos

Todos os códigos, que serão descritos na íntegra nas próximas páginas, podem ser conferidos pelo repositório no Github em https://github.com/JGuerra97/TrabalhoBD2.

Lista de *triggers*

Lista de funções

Lista de códigos auxiliares

4.1 Criando as tabelas

```
DROP TABLE IF EXISTS funcionario CASCADE;
CREATE TABLE funcionario(
  id INTEGER NOT NULL,
  nome VARCHAR(30),
  nivel_permissao INTEGER DEFAULT 1,
  CONSTRAINT funcionario_pk
    PRIMARY KEY(id)
);

DROP TABLE IF EXISTS equipe CASCADE;
CREATE TABLE equipe(
  id INTEGER NOT NULL,
  lider_id INTEGER NOT NULL,
  CONSTRAINT equipe_pk
    PRIMARY KEY(id),
  CONSTRAINT equipe_funcionario_fk
    FOREIGN KEY (lider_id)
    REFERENCES funcionario (id)
    ON DELETE RESTRICT ON UPDATE CASCADE
);

DROP TABLE IF EXISTS equipes_funcionarios CASCADE;
CREATE TABLE equipes_funcionarios (
  funcionario_id INTEGER,
  equipe_id INTEGER,
  CONSTRAINT EQUIPES_FUNCIONARIOS_PK
    PRIMARY KEY (funcionario_id, equipe_id),
  CONSTRAINT EQUIPES_FUNCIONARIOS_FUNCIONARIO_FK
    FOREIGN KEY (funcionario_id)
    REFERENCES funcionario (id) ON DELETE CASCADE,
  CONSTRAINT EQUIPES_FUNCIONARIOS_EQUIPE_FK
    FOREIGN KEY (equipe_id)
    REFERENCES  equipe (id) ON DELETE CASCADE
);

DROP TABLE IF EXISTS categoria CASCADE;
CREATE TABLE categoria(
  nome VARCHAR(20) NOT NULL,
  permissao_assoc INTEGER NOT NULL,
  CONSTRAINT categoria_pk
    PRIMARY KEY(nome)
);
```

```
DROP TABLE IF EXISTS projeto CASCADE;
CREATE TABLE projeto(
  id INTEGER NOT NULL,
  categoria_nome VARCHAR(20) NOT NULL,
  equipe_id INTEGER NOT NULL,
  CONSTRAINT projeto_pk PRIMARY KEY(id),
  CONSTRAINT projeto_categoria_fk
    FOREIGN KEY (categoria_nome)
    REFERENCES categoria(nome)
    ON DELETE RESTRICT
    ON UPDATE CASCADE,
  CONSTRAINT projeto_equipe_fk
    FOREIGN KEY (equipe_id)
    REFERENCES equipe(id)
    ON DELETE RESTRICT
    ON UPDATE CASCADE
);
```

Código auxiliar 1: Criação das tabelas do banco de dados

4.2 Inserções para exemplos e testes

```sql
INSERT INTO funcionario (id, nome, nivel_permissao) VALUES
    (1, 'Rodolfo', 4),
    (2, 'Rosislene', 3),
    (3, 'Mario', 2),
    (4, 'Andreia', 6),
    (5, 'Leonardo', 3);

INSERT INTO equipe (id, lider_id) VALUES
    (1, 1),
    (2, 1),
    (3, 4);

INSERT INTO equipes_funcionarios (funcionario_id, equipe_id) VALUES
    (1, 1),
    (2, 1),
    (3, 1),
    (1, 2),
    (2, 2),
    (4, 3);

INSERT INTO categoria (nome, permissao_assoc) VALUES
    ('Administrativo', 2),
    ('Vendas', 3),
    ('Confidencial', 6),
    ('Marketing', 3);

INSERT INTO projeto (id, categoria_nome, equipe_id) VALUES
    (1, 'Administrativo', 1),
    (2, 'Vendas', 2),
    (3, 'Marketing', 3);
```

Código auxiliar 2: Inserções utilizadas nos exemplos

4.3 Primeira restrição

```
CREATE OR REPLACE FUNCTION altera_categoria_restricao_um_function()
RETURNS TRIGGER AS $$
  DECLARE
        cursor1Restricao1 CURSOR FOR
      SELECT min(nivel_permissao) AS permissao
          FROM funcionario
          JOIN equipes_funcionarios
          ON funcionario.id = equipes_funcionarios.funcionario_id
          JOIN projeto
          ON projeto.equipe_id = equipes_funcionarios.equipe_id
          WHERE projeto.categoria_nome = NEW.nome
          GROUP BY equipes_funcionarios.equipe_id;
  BEGIN
        IF NEW.permissao_assoc > OLD.permissao_assoc THEN
          FOR recebe_cursor IN cursor1Restricao1 LOOP
                IF recebe_cursor.permissao < NEW.permissao_assoc THEN
                  RAISE EXCEPTION 'A permissão de uma equipe
                                   associada a um projeto desta
                                   categoria não é compatível.';
                END IF;
          END LOOP;
    END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;
```

Função 1: [1º restrição] Função do *trigger* em categoria

```
DROP TRIGGER IF EXISTS altera_categoria_restricao_um ON categoria;
CREATE TRIGGER altera_categoria_restricao_um BEFORE UPDATE ON categoria
FOR EACH ROW EXECUTE PROCEDURE altera_categoria_restricao_um_function();
```

Trigger 1: [1º restrição] *Trigger* em categoria

```
CREATE OR REPLACE FUNCTION altera_ou_insere_projeto_restricao_um_function()
RETURNS TRIGGER AS $$
  DECLARE
        permissao_equipe INTEGER;
        permissao_categoria INTEGER;
  BEGIN
        SELECT min(nivel_permissao)
          FROM funcionario
          JOIN equipes_funcionarios
          ON equipes_funcionarios.funcionario_id = funcionario.id
          WHERE equipes_funcionarios.equipe_id = NEW.equipe_id
          GROUP BY equipes_funcionarios.equipe_id
          INTO permissao_equipe;
        SELECT permissao_assoc
          FROM categoria
          WHERE categoria.nome = NEW.categoria_nome
          INTO permissao_categoria;
        IF permissao_equipe < permissao_categoria THEN
          RAISE EXCEPTION 'A permissão associada a categoria
                           deste projeto não é compatível
                           com a da equipe.';
        END IF;
        RETURN NEW;
  END;
$$ LANGUAGE plpgsql;
```

Função 2: [1º restrição] Função do *trigger* em projeto

```
DROP TRIGGER IF EXISTS altera_ou_insere_projeto_restricao_um ON projeto;
CREATE TRIGGER altera_ou_insere_projeto_restricao_um
BEFORE UPDATE OR INSERT ON projeto
FOR EACH ROW EXECUTE
PROCEDURE altera_ou_insere_projeto_restricao_um_function();
```

Trigger 2: [1º restrição] *Trigger* em projeto

```
CREATE OR REPLACE FUNCTION altera_funcionario_restricao_um_function()
RETURNS TRIGGER AS $$
  DECLARE
        cursor1Restricao1 CURSOR FOR
          SELECT max(categoria.permissao_assoc) AS permissaoNecessaria
          FROM projeto
          JOIN categoria
          ON projeto.categoria_nome = categoria.nome
          JOIN equipes_funcionarios
          ON projeto.equipe_id = equipes_funcionarios.equipe_id
          WHERE equipes_funcionarios.funcionario_id = NEW.id
          GROUP BY projeto.equipe_id;
  BEGIN
    FOR maxima_permissao_equipes IN cursor1Restricao1 LOOP
          IF maxima_permissao_equipes.permissaoNecessaria
                                        > NEW.nivel_permissao THEN
                RAISE EXCEPTION 'Ao menos uma equipe deste funcionário
                                 contém projetos cuja permissao necessária
                                 é maior que a do funcionario.';
          END IF;
    END LOOP;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;
```

Função 3: [1º restrição] Função do *trigger* em funcionário

```
DROP TRIGGER IF EXISTS altera_funcionario_restricao_um ON funcionario;
CREATE TRIGGER altera_funcionario_restricao_um BEFORE UPDATE ON funcionario
FOR EACH ROW EXECUTE PROCEDURE altera_funcionario_restricao_um_function();
```

Trigger 3: [1º restrição] *Trigger* em funcionário

```
CREATE OR REPLACE FUNCTION altera_equipe_restricao_um_function()
RETURNS TRIGGER AS $$
  DECLARE
        permissao_equipe INTEGER;
        permissao_lider INTEGER;
  BEGIN
        SELECT max(categoria.permissao_assoc) AS permissao
          FROM projeto JOIN categoria
          ON projeto.categoria_nome = categoria.nome
          WHERE projeto.equipe_id = NEW.id
          INTO permissao_equipe;
        SELECT nivel_permissao
          FROM funcionario
          WHERE funcionario.id = NEW.lider_id
          INTO permissao_lider;
        IF permissao_lider < permissao_equipe THEN
           RAISE EXCEPTION 'O líder não tem permissão para participar
                            dos projetos desta equipe.';
        END IF;
        RETURN NEW;
  END;
$$ LANGUAGE plpgsql;
```

Função 4: [1º restrição] Função do *trigger* em equipe

```
DROP TRIGGER IF EXISTS altera_equipe_restricao_um ON equipe;
CREATE TRIGGER altera_equipe_restricao_um BEFORE UPDATE ON equipe
FOR EACH ROW EXECUTE PROCEDURE altera_equipe_restricao_um_function();
```

Trigger 4: [1º restrição] *Trigger* em equipe

```
CREATE OR REPLACE FUNCTION
    altera_ou_insere_equipes_funcionarios_restricao_um_function()
RETURNS TRIGGER AS $$
  DECLARE
        permissao_projetos_equipe RECORD;
        permissao_funcionario INTEGER;
  BEGIN
        SELECT max(categoria.permissao_assoc) AS permissao,
               count(*) AS qtdProjetos
          FROM projeto
          JOIN categoria
          ON projeto.categoria_nome = categoria.nome
          WHERE projeto.equipe_id = NEW.equipe_id
          GROUP BY projeto.equipe_id
          INTO permissao_projetos_equipe;
        SELECT nivel_permissao
          FROM funcionario
          WHERE funcionario.id = NEW.funcionario_id
          INTO permissao_funcionario;
        IF permissao_projetos_equipe.qtdProjetos < 1 THEN
          RETURN NEW;
        END IF;
        IF permissao_funcionario
                        < permissao_projetos_equipe.permissao THEN
          RAISE EXCEPTION 'O funcionário não tem permissão
                           para os projetos da equipe.';
        END IF;
        RETURN NEW;
  END;
$$ LANGUAGE plpgsql;
```

Função 5: [1º restrição] Função do *trigger* em equipes_funcionarios

```
DROP TRIGGER IF EXISTS altera_ou_insere_equipes_funcionarios_restricao_um
ON equipes_funcionarios;
CREATE TRIGGER altera_ou_insere_equipes_funcionarios_restricao_um
BEFORE UPDATE OR INSERT ON equipes_funcionarios
FOR EACH ROW EXECUTE
PROCEDURE altera_ou_insere_equipes_funcionarios_restricao_um_function();
```

Trigger 5: [1º restrição] *Trigger* em equipes_funcionarios

4.4 Segunda restrição

```
CREATE OR REPLACE FUNCTION verifica_situacao_funcionario_equipe
(id_funcionario INTEGER, permissao_funcionario INTEGER, id_equipe INTEGER)
RETURNS boolean AS $$
  DECLARE
    lider_equipe_id INTEGER;
    maxima_permissao_equipe INTEGER;
    permissao_lider INTEGER;
  BEGIN
    SELECT lider_id
      FROM equipe
      WHERE equipe.id = id_equipe
      INTO lider_equipe_id;
    IF lider_equipe_id = id_funcionario THEN
      SELECT max(nivel_permissao) AS permissao
        FROM funcionario
        JOIN equipes_funcionarios
        ON funcionario.id = equipes_funcionarios.funcionario_id
        WHERE equipes_funcionarios.equipe_id = id_equipe
        AND funcionario.id <> id_funcionario
        GROUP BY equipes_funcionarios.equipe_id
        INTO maxima_permissao_equipe;
      IF(maxima_permissao_equipe >= permissao_funcionario) THEN
        RETURN FALSE;
      END IF;
      RETURN TRUE;
    ELSE
      SELECT nivel_permissao AS permissao
        FROM funcionario
        WHERE funcionario.id = lider_equipe_id
        INTO permissao_lider;
      IF (permissao_lider <= permissao_funcionario) THEN
        RETURN FALSE;
      END IF;
      RETURN TRUE;
    END IF;
  END;
$$ LANGUAGE plpgsql;
```

Código auxiliar 3: Função de apoio para a Segunda Restrição

```
CREATE OR REPLACE FUNCTION altera_funcionario_restricao_dois_function()
RETURNS TRIGGER AS $$
  DECLARE
    cursor1Restricao2 CURSOR FOR
      SELECT equipes_funcionarios.equipe_id
      FROM equipes_funcionarios
      WHERE equipes_funcionarios.funcionario_id = NEW.id;
  BEGIN
    FOR id_equipe IN cursor1Restricao2 LOOP
      IF NOT verifica_situacao_funcionario_equipe(NEW.id, NEW.nivel_permissao,
      id_equipe.equipe_id) THEN
        RAISE EXCEPTION 'Todo lider de equipe deve ter permissao superior
                         aos demais funcionarios da equipe que lidera.';
      END IF;
    END LOOP;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;
```

Função 6: [2º restrição] Função do *trigger* em funcionario

```
DROP TRIGGER IF EXISTS altera_funcionario_restricao_dois ON funcionario;
CREATE TRIGGER altera_funcionario_restricao_dois BEFORE UPDATE ON funcionario
FOR EACH ROW EXECUTE PROCEDURE altera_funcionario_restricao_dois_function();
```

Trigger 6: [2º restrição] *Trigger* em funcionario

```
CREATE OR REPLACE FUNCTION altera_equipe_restricao_dois_function()
RETURNS TRIGGER AS $$
  DECLARE
    permissao_lider_equipe INTEGER;
  BEGIN
    SELECT nivel_permissao
      FROM funcionario
      WHERE funcionario.id = NEW.lider_id
      INTO permissao_lider_equipe;
    IF NOT verifica_situacao_funcionario_equipe(NEW.lider_id,
    permissao_lider_equipe, NEW.id) THEN
      RAISE EXCEPTION 'O lider não tem permissão superior a todos os
                                        funcionários desta equipe.';
    END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;
```

Função 7: [2º restrição] Função do *trigger* em equipe

```
DROP TRIGGER IF EXISTS altera_equipe_restricao_dois ON equipe;
CREATE TRIGGER altera_equipe_restricao_dois AFTER UPDATE ON equipe
FOR EACH ROW EXECUTE PROCEDURE altera_equipe_restricao_dois_function();
```

Trigger 7: [2º restrição] *Trigger* em equipe

```
CREATE OR REPLACE FUNCTION altera_equipes_funcionarios_restricao_dois_function()
RETURNS TRIGGER AS $$
  DECLARE
    permissao_funcionario INTEGER;
  BEGIN
    SELECT nivel_permissao
      FROM funcionario
      WHERE funcionario.id = NEW.funcionario_id
      INTO permissao_funcionario;
    IF NOT verifica_situacao_funcionario_equipe(NEW.funcionario_id,
    permissao_funcionario, NEW.equipe_id) THEN
      RAISE EXCEPTION 'O funcionário da equipe não deve ter permissão superior
                                                    ao lider da equipe.';
    END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;
```

Função 8: [2º restrição] Função do *trigger* em equipes_funcionarios

```
DROP TRIGGER IF EXISTS altera_equipes_funcionarios_restricao_dois
  ON equipes_funcionarios;
CREATE TRIGGER altera_equipes_funcionarios_restricao_dois
  BEFORE UPDATE OR INSERT ON equipes_funcionarios
FOR EACH ROW EXECUTE PROCEDURE
  altera_equipes_funcionarios_restricao_dois_function();
```

Trigger 8: [2º restrição] *Trigger* em equipes_funcionarios

4.5 Terceira restrição

```
CREATE OR REPLACE FUNCTION verifica_somatorio_das_permissoes(id_equipe INTEGER)
RETURNS boolean AS $$
  DECLARE
    somatorio_permissoes_projetos INTEGER;
    somatorio_permissao_equipe INTEGER;
  BEGIN
    IF coalesce(id_equipe, -1) = -1 THEN
      RETURN TRUE;
    END IF;
    SELECT sum(nivel_permissao)
      FROM funcionario
      JOIN equipes_funcionarios
      ON funcionario.id = equipes_funcionarios.funcionario_id
      WHERE equipes_funcionarios.equipe_id = id_equipe
      GROUP BY equipes_funcionarios.equipe_id
      INTO somatorio_permissao_equipe;
    SELECT sum(categoria.permissao_assoc)
      FROM projeto
      JOIN categoria
      ON projeto.categoria_nome = categoria.nome
      WHERE projeto.equipe_id = id_equipe
      INTO somatorio_permissoes_projetos;
    IF somatorio_permissoes_projetos <= (somatorio_permissao_equipe/2) THEN
      RETURN TRUE;
    END IF;
    RETURN FALSE;
  END;
$$ LANGUAGE plpgsql;
```

Código auxiliar 4: Função de apoio para a Terceira Restrição

```
CREATE OR REPLACE FUNCTION altera_categoria_restricao_tres_function()
RETURNS TRIGGER AS $$
  DECLARE
    cursor1Restricao3 CURSOR FOR
      SELECT equipe.id
      FROM equipe
      JOIN projeto
      ON projeto.equipe_id = equipe.id
      JOIN categoria
      ON projeto.categoria_nome = categoria.nome
      WHERE categoria.nome = NEW.nome;
  BEGIN
    IF NEW.permissao_assoc <> OLD.permissao_assoc THEN
      FOR equipe_linha IN cursor1Restricao3 LOOP
        IF NOT verifica_somatorio_das_permissoes(equipe_linha.id) THEN
          RAISE EXCEPTION 'Ao menos uma equipe tem ao menos um projeto desta
          categoria e não tem permissão suficiente para manter esta atualização';
        END IF;
      END LOOP;
    END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;
```

Função 9: [3º restrição] Função do *trigger* em categoria

```
DROP TRIGGER IF EXISTS altera_categoria_restricao_tres ON categoria;
CREATE TRIGGER altera_categoria_restricao_tres AFTER UPDATE ON categoria
FOR EACH STATEMENT EXECUTE PROCEDURE
  altera_categoria_restricao_tres_function();
```

Trigger 9: [3º restrição] *Trigger* em categoria

```
CREATE OR REPLACE FUNCTION altera_ou_insere_projeto_restricao_tres_function()
RETURNS TRIGGER AS $$
  BEGIN
    IF NOT verifica_somatorio_das_permissoes(NEW.equipe_id) THEN
      RAISE EXCEPTION 'A equipe não tem permissão suficiente para assumir
                                                          este projeto.';
    END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;
```

Função 10: [3º restrição] Função do *trigger* em projeto

```
DROP TRIGGER IF EXISTS altera_ou_insere_projeto_restricao_tres ON projeto;
CREATE TRIGGER altera_ou_insere_projeto_restricao_tres
  AFTER UPDATE OR INSERT ON projeto
FOR EACH STATEMENT EXECUTE PROCEDURE
  altera_ou_insere_projeto_restricao_tres_function();
```

Trigger 10: [3º restrição] *Trigger* em projeto

```
CREATE OR REPLACE FUNCTION altera_funcionario_restricao_tres_function()
RETURNS TRIGGER AS $$
  DECLARE
    cursor1Restricao3 CURSOR FOR
      SELECT equipes_funcionarios.equipe_id
      FROM equipes_funcionarios
      WHERE equipes_funcionarios.funcionario_id = NEW.id;
  BEGIN
    IF NEW.nivel_permissao < OLD.nivel_permissao THEN
      FOR id_equipe IN cursor1Restricao3 LOOP
        IF NOT verifica_somatorio_das_permissoes(id_equipe) THEN
          RAISE EXCEPTION 'Alguma das equipes deste funcionario não terá
                    permissão suficiente para os projetos que gerencia.';
        END IF;
      END LOOP;
    END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;
```

Função 11: [3º restrição] Função do *trigger* em funcionario

```
DROP TRIGGER IF EXISTS altera_funcionario_restricao_tres ON funcionario;
CREATE TRIGGER altera_funcionario_restricao_tres
AFTER UPDATE ON funcionario
FOR EACH STATEMENT EXECUTE PROCEDURE
  altera_funcionario_restricao_tres_function();
```

Trigger 11: [3º restrição] *Trigger* em funcionario

```
CREATE OR REPLACE FUNCTION
altera_ou_remove_equipes_funcionarios_restricao_tres_function()
RETURNS TRIGGER AS $$
  BEGIN
    IF NOT verifica_somatorio_das_permissoes(OLD.equipe_id) THEN
      RAISE EXCEPTION 'A equipe não terá perimssão suficiente para manter
                                                os projetos que gerencia.';
    END IF;
    RETURN NEW;
  END;
$$ LANGUAGE plpgsql;
```

Função 12: [3º restrição] Função do *trigger* em equipes_funcionarios

```
DROP TRIGGER IF EXISTS altera_ou_remove_equipes_funcionarios_restricao_tres
  ON equipes_funcionarios;
CREATE TRIGGER altera_ou_insere_remove_funcionarios_restricao_tres
  AFTER UPDATE OR DELETE ON equipes_funcionarios
FOR EACH STATEMENT EXECUTE PROCEDURE
  altera_ou_remove_equipes_funcionarios_restricao_tres_function();
```

Trigger 12: [3º restrição] *Trigger* em equipes_funcionarios

4.6 Primeira função

```
DROP FUNCTION IF EXISTS relatorio_funcionario_equipe;
CREATE OR REPLACE FUNCTION sugestoes_nova_equipe_projeto
                    (permissao_desejada INTEGER, limite_projetos INTEGER)
RETURNS TABLE (id INTEGER,
               nome VARCHAR(30),
               no_projetos BIGINT,
               nivel_permissao INTEGER)
AS $$
  BEGIN
    RETURN QUERY WITH
            t1 AS (SELECT funcionario.id,
           funcionario.nome,
           count(projeto.id) AS no_projetos,
           funcionario.nivel_permissao
           FROM projeto
           JOIN equipe ON projeto.equipe_id = equipe.id
           JOIN equipes_funcionarios
           ON equipe.id = equipes_funcionarios.equipe_id
           JOIN funcionario
           ON equipes_funcionarios.funcionario_id = funcionario.id
           WHERE funcionario.nivel_permissao >= permissao_desejada
           GROUP BY funcionario.id)
        SELECT *
        FROM t1
        WHERE t1.no_projetos < limite_projetos
        ORDER BY t1.no_projetos ASC, nivel_permissao ASC;
  END;
$$ LANGUAGE plpgsql;
```

Função 13: Primeira função

4.7 Segunda função

```
DROP FUNCTION IF EXISTS relatorio_funcionario_equipe;
CREATE OR REPLACE FUNCTION relatorio_funcionario_equipe()
RETURNS TABLE(equipe INTEGER, funcionario VARCHAR(30), cargo VARCHAR(15)) AS $$
  DECLARE
    cursorFuncionarioEquipe CURSOR FOR
      SELECT equipe.id AS eq_id,
      equipe.lider_id AS eq_lider_id,
      funcionario.id AS func_id,
      funcionario.nome AS func_nome,
      equipes_funcionarios.funcionario_id AS eq_func_func_id,
      equipes_funcionarios.equipe_id AS eq_func_eq_id
      FROM (funcionario
      CROSS JOIN equipe)
      LEFT OUTER JOIN equipes_funcionarios
      ON equipes_funcionarios.funcionario_id = funcionario.id
      AND equipes_funcionarios.equipe_id = equipe.id;
  BEGIN
    DROP TABLE IF EXISTS log_funcionarios_equipes;
    CREATE TEMPORARY TABLE log_funcionarios_equipes (
      equipe INTEGER,
      funcionario VARCHAR(30),
      cargo VARCHAR(15),
      PRIMARY KEY (funcionario, equipe)
    );
    FOR linha IN cursorFuncionarioEquipe LOOP
      IF linha.eq_lider_id = linha.func_id THEN
        INSERT INTO log_funcionarios_equipes VALUES
          (linha.eq_id, concat(linha.func_nome, ' (', linha.func_id, ')'),
            'Líder');
      ELSIF coalesce(linha.eq_func_eq_id, -1) <> -1 THEN
        INSERT INTO log_funcionarios_equipes VALUES
          (linha.eq_id, concat(linha.func_nome, ' (', linha.func_id, ')'),
            'Membro');
      ELSE
        INSERT INTO log_funcionarios_equipes VALUES
          (linha.eq_id, concat(linha.func_nome, ' (', linha.func_id, ')'),
            'Não participa');
      END IF;
    END LOOP;
    RETURN QUERY SELECT * FROM log_funcionarios_equipes ORDER BY equipe;
  END;
$$ LANGUAGE plpgsql;
```

Função 14: Segunda função

5 Testes

Lista de testes

5.1 Primeira Restrição

```
UPDATE categoria SET permissao_assoc = 4 WHERE nome = 'Administrativo';
```

ERROR: A permissão de uma equipe associada a um projeto desta categoria não é compatível.

Teste 1: [1º restrição] Teste de atualização para *Trigger* em categoria

```
UPDATE projeto SET categoria_nome = 'Vendas' WHERE id = 1;
```

ERROR: A permissão associada a categoria deste projeto não é compatível com a da equipe.

Teste 2: [1º restrição] Teste de atualização para *Trigger* em projeto

```
INSERT INTO projeto (id, categoria_nome, equipe_id) VALUES (4, 'Vendas', 1);
```

ERROR: A permissão associada a categoria deste projeto não é compatível com a da equipe.

Teste 3: [1º restrição] Teste de inserção para *Trigger* em projeto

```
UPDATE funcionario SET nivel_permissao = 1 WHERE id = 2;
```

ERROR: Ao menos uma equipe deste funcionário contém projetos cuja permissao necessária é maior que a do funcionario.

Teste 4: [1º restrição] Teste de atualização para *Trigger* em funcionario

```
UPDATE equipe SET lider_id = 3 WHERE id = 1;
```

ERROR: O lider não tem permissão superior a todos os funcionários desta equipe.

Teste 5: [1º restrição] Teste de atualização para *Trigger* em equipe

```
INSERT INTO equipes_funcionarios VALUES (3, 2);
```

ERROR: O funcionário não tem permissão para os projetos da equipe.

Teste 6: [1º restrição] Teste de inserção para *Trigger* em equipes_funcionarios

```
UPDATE equipes_funcionarios SET funcionario_id = 3 WHERE funcionario_id = 4;
```

ERROR: O funcionário não tem permissão para os projetos da equipe.

Teste 7: [1º restrição] Teste de atualização para *Trigger* em equipes_funcionarios

5.2 Segunda Restrição

```
UPDATE funcionario SET nivel_permissao = 2 WHERE id = 1;
```

ERROR: Todo lider de equipe deve ter permissao superior aos demais funcionarios da equipe que lidera.

Teste 8: [2º restrição] Teste de atualização para *Trigger* em funcionario

```
UPDATE equipe SET lider_id = 2 WHERE id = 1;
```

ERROR: O lider não tem permissão superior a todos os funcionários desta equipe.

Teste 9: [2º restrição] Teste de atualização para *Trigger* em equipe

```
UPDATE equipes_funcionarios SET funcionario_id = 4
  WHERE equipe_id = 1 AND funcionario_id = 2;
```

ERROR: O funcionário da equipe não deve ter permissão superior ao lider da equipe.

Teste 10: [2º restrição] Teste de atualização para *Trigger* em equipes_funcionarios

```
INSERT INTO equipes_funcionarios VALUES (4, 1);
```

ERROR: O funcionário da equipe não deve ter permissão superior ao lider da equipe.

Teste 11: [2º restrição] Teste de inserção para *Trigger* em equipes_funcionarios

5.3 Terceira Restrição

```
UPDATE categoria SET permissao_assoc = 4 WHERE nome = 'Marketing';
```

ERROR: Ao menos uma equipe tem ao menos um projeto desta categoria e não tem permissão suficiente para manter esta atualização.

Teste 12: [3º restrição] Teste de atualização para *Trigger* em categoria

```
INSERT INTO projeto VALUES (4, 'Administrativo', 3);
```

ERROR: A equipe não tem permissão suficiente para assumir este projeto.

Teste 13: [3º restrição] Teste de inserção para *Trigger* em projeto

```
UPDATE projeto SET categoria_nome = 'Confidencial' WHERE id = 3;
```

ERROR: A equipe não tem permissão suficiente para assumir este projeto.

Teste 14: [3º restrição] Teste de atualização para *Trigger* em projeto

```
UPDATE funcionario SET nivel_permissao = 5 WHERE id = 4;
```

ERROR: Alguma das equipes deste funcionario não terá permissão suficiente para os projetos que gerencia.

Teste 15: [3º restrição] Teste de atualização para *Trigger* em funcionario

```
UPDATE equipes_funcionarios SET funcionario_id = 5
  WHERE equipe_id = 3 AND funcionario_id = 4;
```

ERROR: A equipe não terá perimssão suficiente para manter os projetos que gerencia.

Teste 16: [3º restrição] Teste de atualização para *Trigger* em equipes_funcionarios

```sql
DELETE FROM equipes_funcionarios WHERE funcionario_id = 2 AND equipe_id = 2;
```

ERROR: A equipe não terá perimssão suficiente para manter os projetos que gerencia.

Teste 17: [3º restrição] Teste de remoção para *Trigger* em equipes_funcionarios