```
-- ====================================================
-- FUNÇÕES E TRIGGERS - RESTRIÇÕES DE INTEGRIDADE
-- ====================================================

-- ====================================================
-- 1. HIERARQUIA DE CONFLITOS (EXCLUSIVA E TOTAL)
-- ====================================================
-- Função para validar exclusividade da hierarquia
CREATE OR REPLACE FUNCTION fn_valida_exclusividade_conflito()
RETURNS TRIGGER AS $$
DECLARE
    cod_conflito_atual INT;
    tipos_existentes INT := 0;
BEGIN
    cod_conflito_atual := NEW.cod_conflito_fk;

    -- Conta quantos tipos já existem para este conflito
    SELECT
        (CASE WHEN EXISTS(SELECT 1 FROM Conflito_Territorial WHERE cod_conflito_fk =
cod_conflito_atual) THEN 1 ELSE 0 END) +
        (CASE WHEN EXISTS(SELECT 1 FROM Conflito_Religioso WHERE cod_conflito_fk =
cod_conflito_atual) THEN 1 ELSE 0 END) +
        (CASE WHEN EXISTS(SELECT 1 FROM Conflito_Economico WHERE cod_conflito_fk
= cod_conflito_atual) THEN 1 ELSE 0 END) +
        (CASE WHEN EXISTS(SELECT 1 FROM Conflito_Racial WHERE cod_conflito_fk =
cod_conflito_atual) THEN 1 ELSE 0 END)
    INTO tipos_existentes;

    IF tipos_existentes >= 1 THEN
        RAISE EXCEPTION 'EXCLUSIVIDADE VIOLADA: Conflito % já possui um tipo
específico. Um conflito só pode ter um tipo.', cod_conflito_atual;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Triggers para exclusividade
CREATE TRIGGER tg_exclusividade_conflito_territorial
    BEFORE INSERT ON Conflito_Territorial
    FOR EACH ROW EXECUTE FUNCTION fn_valida_exclusividade_conflito();

CREATE TRIGGER tg_exclusividade_conflito_religioso
    BEFORE INSERT ON Conflito_Religioso
    FOR EACH ROW EXECUTE FUNCTION fn_valida_exclusividade_conflito();

CREATE TRIGGER tg_exclusividade_conflito_economico
    BEFORE INSERT ON Conflito_Economico
```

```
    FOR EACH ROW EXECUTE FUNCTION fn_valida_exclusividade_conflito();

CREATE TRIGGER tg_exclusividade_conflito_racial
    BEFORE INSERT ON Conflito_Racial
    FOR EACH ROW EXECUTE FUNCTION fn_valida_exclusividade_conflito();

-- Função para validar totalidade (impedir conflito sem tipo)
CREATE OR REPLACE FUNCTION fn_valida_totalidade_conflito()
RETURNS TRIGGER AS $$
DECLARE
    tipos_existentes INT := 0;
BEGIN
    SELECT
        (CASE WHEN EXISTS(SELECT 1 FROM Conflito_Territorial WHERE cod_conflito_fk =
OLD.cod_conflito) THEN 1 ELSE 0 END) +
        (CASE WHEN EXISTS(SELECT 1 FROM Conflito_Religioso WHERE cod_conflito_fk =
OLD.cod_conflito) THEN 1 ELSE 0 END) +
        (CASE WHEN EXISTS(SELECT 1 FROM Conflito_Economico WHERE cod_conflito_fk
= OLD.cod_conflito) THEN 1 ELSE 0 END) +
        (CASE WHEN EXISTS(SELECT 1 FROM Conflito_Racial WHERE cod_conflito_fk =
OLD.cod_conflito) THEN 1 ELSE 0 END)
    INTO tipos_existentes;

    IF tipos_existentes = 0 THEN
        RAISE EXCEPTION 'TOTALIDADE VIOLADA: Conflito % deve ter pelo menos um tipo
específico.', OLD.cod_conflito;
    END IF;

    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER tg_totalidade_conflito
    BEFORE DELETE ON Conflito
    FOR EACH ROW EXECUTE FUNCTION fn_valida_totalidade_conflito();

-- Função para impedir remoção do último tipo
CREATE OR REPLACE FUNCTION fn_impede_remocao_ultimo_tipo()
RETURNS TRIGGER AS $$
DECLARE
    cod_conflito_atual INT;
    tipos_restantes INT := 0;
BEGIN
    cod_conflito_atual := OLD.cod_conflito_fk;

    SELECT
        (CASE WHEN EXISTS(SELECT 1 FROM Conflito_Territorial WHERE cod_conflito_fk =
cod_conflito_atual) AND TG_TABLE_NAME != 'conflito_territorial' THEN 1 ELSE 0 END) +
```

```
        (CASE WHEN EXISTS(SELECT 1 FROM Conflito_Religioso WHERE cod_conflito_fk =
cod_conflito_atual) AND TG_TABLE_NAME != 'conflito_religioso' THEN 1 ELSE 0 END) +
        (CASE WHEN EXISTS(SELECT 1 FROM Conflito_Economico WHERE cod_conflito_fk
= cod_conflito_atual) AND TG_TABLE_NAME != 'conflito_economico' THEN 1 ELSE 0 END)
+
        (CASE WHEN EXISTS(SELECT 1 FROM Conflito_Racial WHERE cod_conflito_fk =
cod_conflito_atual) AND TG_TABLE_NAME != 'conflito_racial' THEN 1 ELSE 0 END)
    INTO tipos_restantes;

    IF tipos_restantes = 0 THEN
        RAISE EXCEPTION 'TOTALIDADE VIOLADA: Não é possível remover o último tipo do
conflito %.', cod_conflito_atual;
    END IF;

    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

-- Triggers para impedir remoção do último tipo
CREATE TRIGGER tg_impede_remocao_ultimo_tipo_territorial
    BEFORE DELETE ON Conflito_Territorial
    FOR EACH ROW EXECUTE FUNCTION fn_impede_remocao_ultimo_tipo();

CREATE TRIGGER tg_impede_remocao_ultimo_tipo_religioso
    BEFORE DELETE ON Conflito_Religioso
    FOR EACH ROW EXECUTE FUNCTION fn_impede_remocao_ultimo_tipo();

CREATE TRIGGER tg_impede_remocao_ultimo_tipo_economico
    BEFORE DELETE ON Conflito_Economico
    FOR EACH ROW EXECUTE FUNCTION fn_impede_remocao_ultimo_tipo();

CREATE TRIGGER tg_impede_remocao_ultimo_tipo_racial
    BEFORE DELETE ON Conflito_Racial
    FOR EACH ROW EXECUTE FUNCTION fn_impede_remocao_ultimo_tipo();

-- ====================================================
-- 2. MÁXIMO 3 CHEFES POR DIVISÃO
-- ====================================================
CREATE OR REPLACE FUNCTION fn_valida_max_chefes_divisao()
RETURNS TRIGGER AS $$
DECLARE
    num_chefes_atuais INT;
BEGIN
    IF NEW.cod_grupo_divisao_liderada_fk IS NOT NULL AND
NEW.num_divisao_liderada_fk IS NOT NULL THEN
        SELECT COUNT(*)
        INTO num_chefes_atuais
        FROM Chefe_Militar
```

```
        WHERE cod_grupo_divisao_liderada_fk = NEW.cod_grupo_divisao_liderada_fk
          AND num_divisao_liderada_fk = NEW.num_divisao_liderada_fk
          AND (TG_OP = 'INSERT' OR cod_chefe != NEW.cod_chefe);

        IF num_chefes_atuais >= 3 THEN
            RAISE EXCEPTION 'Uma divisão não pode ser liderada por mais de 3 chefes
militares. Divisão: (%, %)',
                NEW.cod_grupo_divisao_liderada_fk, NEW.num_divisao_liderada_fk;
        END IF;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER tg_valida_max_chefes_divisao
    BEFORE INSERT OR UPDATE OF cod_grupo_divisao_liderada_fk,
num_divisao_liderada_fk ON Chefe_Militar
    FOR EACH ROW EXECUTE FUNCTION fn_valida_max_chefes_divisao();

-- ====================================================
-- 3. MÍNIMO 2 GRUPOS POR CONFLITO
-- ====================================================
CREATE OR REPLACE FUNCTION fn_valida_min_dois_grupos_conflito()
RETURNS TRIGGER AS $$
DECLARE
    num_grupos_no_conflito INT;
BEGIN
    SELECT COUNT(DISTINCT cod_grupo_fk)
    INTO num_grupos_no_conflito
    FROM Grupo_Armado_Participa_Conflito
    WHERE cod_conflito_fk = OLD.cod_conflito_fk;

    IF (num_grupos_no_conflito - 1) < 2 THEN
        RAISE EXCEPTION 'Um conflito deve ter pelo menos dois grupos armados
participantes. Conflito: %', OLD.cod_conflito_fk;
    END IF;

    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER tg_valida_min_dois_grupos_conflito
    BEFORE DELETE ON Grupo_Armado_Participa_Conflito
    FOR EACH ROW EXECUTE FUNCTION fn_valida_min_dois_grupos_conflito();

-- ====================================================
-- 4. BAIXAS TOTAIS E SEQUENCIALIDADE
```

```
-- ================================================
-- Atualizar baixas totais do grupo
CREATE OR REPLACE FUNCTION fn_atualiza_baixas_grupo_armado()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'DELETE' THEN
        UPDATE Grupo_Armado
        SET num_baixas_total_calculado = COALESCE((SELECT SUM(num_baixas_divisao)
FROM Divisao WHERE cod_grupo_fk = OLD.cod_grupo_fk), 0)
        WHERE cod_grupo = OLD.cod_grupo_fk;
    ELSE
        UPDATE Grupo_Armado
        SET num_baixas_total_calculado = COALESCE((SELECT SUM(num_baixas_divisao)
FROM Divisao WHERE cod_grupo_fk = NEW.cod_grupo_fk), 0)
        WHERE cod_grupo = NEW.cod_grupo_fk;
    END IF;

    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER tg_atualiza_baixas_grupo_armado
    AFTER INSERT OR UPDATE OF num_baixas_divisao OR DELETE ON Divisao
    FOR EACH ROW EXECUTE FUNCTION fn_atualiza_baixas_grupo_armado();

-- Definir número de divisão sequencial
CREATE OR REPLACE FUNCTION fn_define_num_divisao_consecutiva()
RETURNS TRIGGER AS $$
DECLARE
    max_num_divisao INT;
BEGIN
    SELECT COALESCE(MAX(num_divisao), 0)
    INTO max_num_divisao
    FROM Divisao
    WHERE cod_grupo_fk = NEW.cod_grupo_fk;

    NEW.num_divisao = max_num_divisao + 1;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER tg_define_num_divisao_consecutiva
    BEFORE INSERT ON Divisao
    FOR EACH ROW EXECUTE FUNCTION fn_define_num_divisao_consecutiva();

-- ================================================
-- 5. RESTRIÇÕES DE INTEGRIDADE MELHORADAS
```

```
-- ================================================
-- Impedir grupo sem divisão
CREATE OR REPLACE FUNCTION fn_impede_grupo_sem_divisao()
RETURNS TRIGGER AS $$
DECLARE
    num_divisoes_restantes INT;
BEGIN
    -- Conta divisões restantes APÓS a remoção
    SELECT COUNT(*) - 1
    INTO num_divisoes_restantes
    FROM Divisao
    WHERE cod_grupo_fk = OLD.cod_grupo_fk;

    IF num_divisoes_restantes < 1 THEN
        RAISE EXCEPTION 'INTEGRIDADE: Grupo armado deve ter pelo menos uma divisão.
Grupo: %', OLD.cod_grupo_fk;
    END IF;

    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER tg_impede_grupo_sem_divisao
    BEFORE DELETE ON Divisao
    FOR EACH ROW EXECUTE FUNCTION fn_impede_grupo_sem_divisao();

-- Impedir grupo sem líder
CREATE OR REPLACE FUNCTION fn_valida_grupo_tem_lider()
RETURNS TRIGGER AS $$
DECLARE
    num_lideres_restantes INT;
BEGIN
    -- Conta líderes restantes APÓS a remoção
    SELECT COUNT(*) - 1
    INTO num_lideres_restantes
    FROM Lider_Politico
    WHERE cod_grupo_liderado_fk = OLD.cod_grupo_liderado_fk;

    IF num_lideres_restantes < 1 THEN
        RAISE EXCEPTION 'INTEGRIDADE: Grupo armado deve ter pelo menos um líder
político. Grupo: %', OLD.cod_grupo_liderado_fk;
    END IF;

    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER tg_valida_grupo_tem_lider
```

```
    BEFORE DELETE ON Lider_Politico
    FOR EACH ROW EXECUTE FUNCTION fn_valida_grupo_tem_lider();

-- ====================================================
-- 6. RESTRIÇÃO DE CHEFE MILITAR COM DIVISÃO DO MESMO GRUPO
-- ====================================================
CREATE OR REPLACE FUNCTION fn_valida_consistencia_grupo_chefe_divisao()
RETURNS TRIGGER AS $$
DECLARE
    v_cod_grupo_do_lider INT;
    v_cod_grupo_da_divisao INT;
BEGIN
    -- Se não há divisão associada, não há o que validar
    IF NEW.cod_grupo_divisao_liderada_fk IS NULL THEN
        RETURN NEW;
    END IF;

    -- Busca o grupo que o líder político (ao qual o chefe obedece) lidera
    SELECT cod_grupo_liderado_fk
    INTO v_cod_grupo_do_lider
    FROM Lider_Politico
    WHERE id_lider_politico = NEW.id_lider_politico_obedece_fk;

    -- O grupo da divisão que o chefe vai liderar
    v_cod_grupo_da_divisao := NEW.cod_grupo_divisao_liderada_fk;

    IF v_cod_grupo_do_lider != v_cod_grupo_da_divisao THEN
        RAISE EXCEPTION 'INCONSISTÊNCIA: Chefe militar só pode liderar divisão do
mesmo grupo de seu líder político. Grupo do líder: %, Grupo da divisão: %',
            v_cod_grupo_do_lider, v_cod_grupo_da_divisao;
    END IF;

    -- Se chegou aqui, a operação é válida
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER tg_valida_consistencia_grupo_chefe_divisao
    BEFORE INSERT OR UPDATE ON Chefe_Militar
    FOR EACH ROW EXECUTE FUNCTION fn_valida_consistencia_grupo_chefe_divisao();

-- ====================================================
-- 7. VALIDAÇÕES DE DATAS
-- ====================================================
-- Validação de datas em participação de grupos em conflitos
CREATE OR REPLACE FUNCTION fn_valida_datas_participacao_grupo()
RETURNS TRIGGER AS $$
BEGIN
```

```
    IF NEW.data_saida IS NOT NULL AND NEW.data_saida < NEW.data_incorporacao
THEN
        RAISE EXCEPTION 'Data de saída não pode ser anterior à data de incorporação';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER tg_valida_datas_participacao_grupo
    BEFORE INSERT OR UPDATE ON Grupo_Armado_Participa_Conflito
    FOR EACH ROW EXECUTE FUNCTION fn_valida_datas_participacao_grupo();

-- Validação de datas em intervenção de organizações
CREATE OR REPLACE FUNCTION fn_valida_datas_intervencao_org()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.data_saida IS NOT NULL AND NEW.data_saida < NEW.data_incorporacao
THEN
        RAISE EXCEPTION 'Data de saída não pode ser anterior à data de incorporação';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER tg_valida_datas_intervencao_org
    BEFORE INSERT OR UPDATE ON Organizacao_Intervem_Conflito
    FOR EACH ROW EXECUTE FUNCTION fn_valida_datas_intervencao_org();

-- ====================================================
-- 8. VALIDAR ESTOQUE DE ARMAS
-- ====================================================
-- Validar se há estoque suficiente antes do fornecimento
CREATE OR REPLACE FUNCTION fn_valida_estoque_armas()
RETURNS TRIGGER AS $$
DECLARE
    estoque_atual INT;
BEGIN
    -- Busca estoque atual
    SELECT quantidade_disponivel
    INTO estoque_atual
    FROM Traficante_Dispoe_Tipo_Arma
    WHERE id_traficante_fk = NEW.id_traficante_fk
      AND nome_arma_fk = NEW.nome_arma_fk;

    -- Verifica se há estoque suficiente
    IF estoque_atual IS NULL OR estoque_atual < NEW.quantidade_fornecida THEN
```

```
        RAISE EXCEPTION 'ESTOQUE INSUFICIENTE: Traficante % não possui estoque
suficiente de %. Estoque: %, Solicitado: %',
            NEW.id_traficante_fk, NEW.nome_arma_fk, COALESCE(estoque_atual, 0),
NEW.quantidade_fornecida;
    END IF;

    -- Reduz o estoque
    UPDATE Traficante_Dispoe_Tipo_Arma
    SET quantidade_disponivel = quantidade_disponivel - NEW.quantidade_fornecida
    WHERE id_traficante_fk = NEW.id_traficante_fk
      AND nome_arma_fk = NEW.nome_arma_fk;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER tg_valida_estoque_armas
    BEFORE INSERT ON Fornecimento_Arma_Grupo
    FOR EACH ROW EXECUTE FUNCTION fn_valida_estoque_armas();

-- ====================================================
-- 9. PROCEDIMENTOS PARA CRIAÇÃO CONSISTENTE
-- ====================================================
-- Procedure para criar conflito com tipo específico
CREATE OR REPLACE FUNCTION sp_criar_conflito_com_tipo(
    p_nome_conflito VARCHAR(255),
    p_tipo_conflito VARCHAR(20),
    p_num_mortos INT DEFAULT 0,
    p_num_feridos INT DEFAULT 0
)
RETURNS INT AS $$
DECLARE
    novo_cod_conflito INT;
BEGIN
    IF p_tipo_conflito NOT IN ('territorial', 'religioso', 'economico', 'racial') THEN
        RAISE EXCEPTION 'Tipo de conflito inválido. Use: territorial, religioso, economico ou
racial';
    END IF;

    INSERT INTO Conflito (nome_conflito, num_mortos_atual, num_feridos_atual)
    VALUES (p_nome_conflito, p_num_mortos, p_num_feridos)
    RETURNING cod_conflito INTO novo_cod_conflito;

    IF p_tipo_conflito = 'territorial' THEN
        INSERT INTO Conflito_Territorial (cod_conflito_fk) VALUES (novo_cod_conflito);
    ELSIF p_tipo_conflito = 'religioso' THEN
        INSERT INTO Conflito_Religioso (cod_conflito_fk) VALUES (novo_cod_conflito);
    ELSIF p_tipo_conflito = 'economico' THEN
```

```
        INSERT INTO Conflito_Economico (cod_conflito_fk) VALUES (novo_cod_conflito);
    ELSIF p_tipo_conflito = 'racial' THEN
        INSERT INTO Conflito_Racial (cod_conflito_fk) VALUES (novo_cod_conflito);
    END IF;

    RETURN novo_cod_conflito;
END;
$$ LANGUAGE plpgsql;

-- Procedure para criar grupo armado com líder e divisão
CREATE OR REPLACE FUNCTION sp_criar_grupo_armado_completo(
    p_nome_grupo VARCHAR(255),
    p_nome_lider VARCHAR(255),
    p_apoios_descricao TEXT DEFAULT NULL
)
RETURNS INT AS $$
DECLARE
    novo_cod_grupo INT;
    novo_id_lider INT;
BEGIN
    -- Criar grupo armado
    INSERT INTO Grupo_Armado (nome_grupo)
    VALUES (p_nome_grupo)
    RETURNING cod_grupo INTO novo_cod_grupo;

    -- Criar líder político
    INSERT INTO Lider_Politico (nome_lider, cod_grupo_liderado_fk, apoios_descricao)
    VALUES (p_nome_lider, novo_cod_grupo, p_apoios_descricao)
    RETURNING id_lider_politico INTO novo_id_lider;

    -- Criar primeira divisão (número será definido automaticamente pelo trigger)
    INSERT INTO Divisao (cod_grupo_fk) VALUES (novo_cod_grupo);

    RETURN novo_cod_grupo;
END;
$$ LANGUAGE plpgsql;
```