



## Regras de Negócio

- **Da relação participa**

Verifica a validade da inserção de um novo participante. O número de participantes do evento ainda não excedeu o limite. O formato do deck deve ser o mesmo do formato do evento, e se o deck é um deck válido, ou seja, a quantidade de cartas pertencentes a ele é maior ou igual ao mínimo do formato.

```
CREATE OR REPLACE FUNCTION checa_formato_evento() RETURNS trigger AS $$
DECLARE
    formato_ev integer;
    formato_dk integer;
    limi integer;
    participante integer;
    qtd integer;
    mini integer;
BEGIN
    SELECT formato_id, limite, participantes INTO formato_ev, limi, participante
        FROM eventos
        WHERE eventos.id = NEW.evento_id;
    SELECT formato_id INTO formato_dk
        FROM decks
        WHERE decks.id = NEW.deck_id;
    IF formato_ev != formato_dk THEN
        RAISE EXCEPTION 'formatos invalido % %', formato_ev, formato_dk;
    END IF;
    IF limi = participante THEN
        RAISE EXCEPTION 'limite excedido %', limi;
    END IF;

    SELECT sum(quantidade) INTO qtd
        from pertence
        where deck_id = new.deck_id;
    SELECT minimo AS mi INTO mini
        from formatos
        where id = formato_ev;
    IF (qtd < mini) OR qtd is null then
        RAISE EXCEPTION 'minimo de cartas nao respeitado %/%', qtd, mini;
    END IF;
    RETURN NEW;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER insere_participa_trigger BEFORE INSERT OR UPDATE ON participa
FOR EACH ROW EXECUTE PROCEDURE checa_formato_evento();
```

Após a inserção, atualização ou remoção de um participante atualiza o número de participantes de acordo.

```
CREATE OR REPLACE FUNCTION incrementa_participantes() RETURNS trigger AS $$
DECLARE
BEGIN
    IF TG_OP != 'INSERT' THEN
        UPDATE eventos
        SET participantes = (SELECT count(*) FROM participa WHERE evento_id = OLD.evento_id)
        WHERE id = OLD.evento_id;
    END IF;
    IF TG_OP != 'DELETE' THEN
        UPDATE eventos
        SET participantes = (SELECT count(*) FROM participa WHERE evento_id = NEW.evento_id)
        WHERE id = NEW.evento_id;
    END IF;
    IF TG_OP = 'DELETE' THEN
        RETURN OLD;
    END IF;
    RETURN NEW;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER incrementa_participantes_trigger AFTER INSERT OR UPDATE OR DELETE ON participa
FOR EACH ROW EXECUTE PROCEDURE incrementa_participantes();
```

- **Da tabela eventos**

Garante que um evento que está sendo alterado ou removido não tenha nenhum participante inscrito.

```
CREATE OR REPLACE FUNCTION muda_evento() RETURNS trigger AS $$
DECLARE
    _participante integer;
BEGIN
    if TG_OP = 'UPDATE' then
        if OLD.participantes != 0 and old.formato_id != new.formato_id then
            RAISE EXCEPTION 'participante ja inscritos';
        end if;
    else
        if OLD.participantes != 0 then
            RAISE EXCEPTION 'participante ja inscritos';
        end if;
    end if;
    return new;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER muda_evento_trigger BEFORE UPDATE OR DELETE ON eventos
FOR EACH ROW EXECUTE PROCEDURE muda_evento();
```

- **Da relação pertence**

Garante que a carta que está sendo inserida em um deck é uma carta legal. A quantidade de cartas iguais do formato do deck deve ser respeitada, a não ser que a carta seja de um tipo *basic\_land*. A carta sendo inserida não pertence a lista de cartas banidas de um formato. Se o formato tem um limite de cartas em um deck, só insere se esse limite não for ser estourado.

```
CREATE OR REPLACE FUNCTION checa_validade_carta() RETURNS trigger AS $$
DECLARE
    qtd integer;
    formato integer;
    banida bool;
    land bool;
    maxi integer;
    qtd2 integer;
BEGIN
    SELECT formato_id into formato
    FROM decks
    where id = new.deck_id;

    SELECT basic_land INTO land
    from tipos_cartas
    inner join cartas on tipos_cartas.id = cartas.tipo_id
    where cartas.id = new.carta_id;

    SELECT qtd_repetidas, maximo INTO qtd, maxi
    from formatos
    where id = formato;

    if new.quantidade > qtd and not land then
        RAISE EXCEPTION 'limite excedido %', qtd;
    end if;

    IF EXISTS (SELECT carta_id FROM BANE WHERE formato_id = formato AND carta_id = new.carta_id)
THEN
        RAISE EXCEPTION 'Carta banida no formato do deck';
    END IF;

    IF maxi is not null then
        SELECT sum(quantidade) INTO qtd2
        from pertence
        where deck_id = new.deck_id;
        if (qtd2 + new.quantidade) > maxi then
            RAISE EXCEPTION 'limite excedido %/%', (qtd2+new.quantidade), maxi;
        end if;
    end if;

    return new;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER checa_validade_carta_trigger BEFORE INSERT OR UPDATE ON pertence
FOR EACH ROW EXECUTE PROCEDURE checa_validade_carta();
```

Caso o formato do deck tenha comandante, garante que a primeira e somente a primeira carta inserida no deck seja comandante. Além disso, garante que as próximas cartas inseridas ou pertencerão às cores do comandante, ou serão incolores.

```
CREATE OR REPLACE FUNCTION checa_commander_carta() RETURNS trigger AS $$
DECLARE
    commander bool;
    tem_commander integer;
BEGIN
    SELECT tem_comandante INTO commander
        from formatos
        where id = (SELECT formato_id FROM decks where id = new.deck_id);
    if commander then
        SELECT carta_id INTO tem_commander FROM pertence WHERE deck_id = new.deck_id AND
is_comandante = true;
        IF new.is_comandante and tem_commander is not null THEN
            RAISE EXCEPTION 'Comandante ja comandando';
        ELSIF tem_commander is not null THEN
            IF (EXISTS ((select cor_id from colore where carta_id = new.carta_id) EXCEPT (select
cor_id from colore where carta_id =tem_commander)))
                and not EXISTS (SELECT cor_id from colore inner join cores on (cor_id =
cores.id) where carta_id = new.carta_id and cores.incolor = true)
            then
                RAISE EXCEPTION 'Cor incompativel com o comandante';
            end if;
        ELSIF not new.is_comandante then
            RAISE EXCEPTION 'Esperando comandante';
        END IF;
    END IF;
    return new;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER checa_commander_carta_trigger BEFORE INSERT OR UPDATE ON pertence
FOR EACH ROW EXECUTE PROCEDURE checa_commander_carta();
```

- **Da relação colore**

Garante que se uma carta é incolor então ela não pode ter outras cores, e se uma carta tem alguma cor, então ela não pode ser incolor.

```
CREATE OR REPLACE FUNCTION checa_cor_carta() RETURNS trigger AS $$
DECLARE
    incolor integer[];
    cores integer[];
BEGIN
    incolor := (SELECT array_agg(cor_id)
                from colore inner join cores on (cor_id = cores.id)
                where carta_id = new.carta_id and cores.incolor = true);
    cores := (SELECT array_agg(cor_id)
              from colore inner join cores on (cor_id = cores.id)
              where carta_id = new.carta_id and cores.incolor = false);
    IF array_length(incolor, 1) > 0 and array_length(cores, 1) > 0 THEN
        RAISE EXCEPTION 'Carta nao pode ser incolor e ter cor ao mesmo tempo';
    END IF;
    IF EXISTS (SELECT cor_id
               from colore inner join cores on (cor_id = cores.id)
               where carta_id = new.carta_id and cores.incolor = true) AND
       EXISTS (SELECT id
               from cores
               where id = new.cor_id and cores.incolor = false)
    THEN
        RAISE EXCEPTION 'Carta já é incolor';
    END IF;
    IF EXISTS (SELECT cor_id
               from colore inner join cores on (cor_id = cores.id)
               where carta_id = new.carta_id and cores.incolor = false) AND
       EXISTS (SELECT id
               from cores
               where id = new.cor_id and cores.incolor = true)
    THEN
        RAISE EXCEPTION 'Carta já tem cor';
    END IF;
    return new;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER checa_cor_carta_trigger BEFORE INSERT OR UPDATE ON colore
FOR EACH ROW EXECUTE PROCEDURE checa_cor_carta();
```

## Após Revisão

Após a apresentação foram feitas alterações nos triggers de três regras:

- Formato do deck deve ser o mesmo do formato do evento.

```
CREATE OR REPLACE FUNCTION muda_evento() RETURNS trigger AS $$
DECLARE
    _participante integer;
BEGIN
    if TG_OP = 'UPDATE' then
        if OLD.participantes != 0 and old.formato_id != new.formato_id then
            RAISE EXCEPTION 'participante ja inscitos';
        end if;
    else
        if OLD.participantes != 0 then
            RAISE EXCEPTION 'participante ja inscitos';
        end if;
    end if;
    return new;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER muda_evento_trigger BEFORE UPDATE OR DELETE ON eventos
FOR EACH ROW EXECUTE PROCEDURE muda_evento();

CREATE OR REPLACE FUNCTION muda_deck() RETURNS trigger AS $$
DECLARE
BEGIN
    if EXISTS (SELECT * FROM participa WHERE participa.deck_id = old.id) then
        RAISE EXCEPTION 'Deck participa de algum evento.';
    end if;
    return new;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER muda_deck_trigger BEFORE UPDATE OR DELETE ON decks
FOR EACH ROW EXECUTE PROCEDURE muda_deck();
```

- A quantidade de cartas iguais do formato do deck deve ser respeitada, a não ser que a carta seja de um tipo basic\_land.

```
CREATE OR REPLACE FUNCTION muda_formato() RETURNS trigger AS $$
DECLARE
BEGIN
    if EXISTS (SELECT * FROM decks WHERE decks.formato_id = old.id) then
        RAISE EXCEPTION 'Formato não pode ser alterado se algum deck usa ele.';
    end if;
    return new;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER muda_formato_trigger BEFORE UPDATE OR DELETE ON formatos
FOR EACH ROW EXECUTE PROCEDURE muda_formato();
```

```

CREATE OR REPLACE FUNCTION muda_tipo_carta() RETURNS trigger AS $$
DECLARE
BEGIN
    if EXISTS (SELECT * FROM cartas WHERE cartas.tipo_id = old.id) then
        RAISE EXCEPTION 'Alguma carta já usa tipo.';
    end if;
    return new;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER muda_tipo_carta_trigger BEFORE UPDATE OR DELETE ON tipos_cartas
FOR EACH ROW EXECUTE PROCEDURE muda_tipo_carta();

```

- A carta sendo inserida não pertence a lista de cartas banidas de um formato.

```

CREATE OR REPLACE FUNCTION muda_banida() RETURNS trigger AS $$
DECLARE
BEGIN
    if EXISTS (SELECT * FROM pertence INNER JOIN decks on (pertence.deck_id = decks.id)
        WHERE pertence.carta_id = new.carta_id and decks.formato_id = new.formato_id)
    then
        RAISE EXCEPTION 'Carta a ser banida pertence a um deck.';
    end if;
    return new;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER muda_banida_trigger BEFORE UPDATE OR INSERT ON bane
FOR EACH ROW EXECUTE PROCEDURE muda_banida();

CREATE OR REPLACE FUNCTION muda_formato_deck() RETURNS trigger AS $$
DECLARE
BEGIN
    if EXISTS (SELECT * FROM (SELECT * FROM pertence WHERE deck_id = old.id) AS cards INNER JOIN
    bane on (cards.carta_id = bane.carta_id)
        WHERE bane.formato_id = new.formato_id ) or new.id <> old.id then
        RAISE EXCEPTION 'O deck tem alguma carta banida no novo formato';
    end if;
    if EXISTS (SELECT * FROM pertence WHERE deck_id = old.id and quantidade <= (SELECT
    qtd_repetidas FROM formatos WHERE id = new.formato_id)) then
        RAISE EXCEPTION 'Alguma carta não respeita o limite de quantidade';
    end if;
    return new;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER muda_formato_deck_trigger BEFORE UPDATE ON decks
FOR EACH ROW EXECUTE PROCEDURE muda_formato_deck();

```



## Outras Consultas

- **Inserção de carta**

Procedure que insere carta e já insere no relacionamento de colore.

```
CREATE OR REPLACE FUNCTION insere_carta(_nome text, _tipo_id integer, _raridade text, _texto text,
 _artista_id integer, _colecacao_id integer, VARIADIC cores integer[]) RETURNS void AS $$
DECLARE
    _id integer;
    _cor_id integer;
BEGIN
    _id := (SELECT coalesce(max(id), 0) + 1 FROM cartas);
    INSERT INTO cartas (id, nome, tipo_id, raridade, texto, artista_id, colecao_id)
    VALUES (_id, _nome, _tipo_id, _raridade, _texto, _artista_id, _colecacao_id);
    foreach _cor_id in array cores loop
        INSERT INTO colore (carta_id, cor_id) VALUES (_id, _cor_id);
    end loop;
END; $$ LANGUAGE plpgsql;
```

- **Cartas mais jogadas**

Consulta que retorna as n cartas mais jogadas num evento.

```
CREATE OR REPLACE FUNCTION cartas_mais_jogadas(_evento integer, n_ranking integer)
RETURNS TABLE( carta integer,
                aparicoes bigint
                ) AS $$
DECLARE
BEGIN
    return QUERY SELECT carta_id, sum(quantidade) as aparicoes
                  from participa inner join pertence on (participa.deck_id = pertence.deck_id and
participa.evento_id = _evento)
                  where carta_id != (select cartas.id
                                     from cartas inner join tipos_cartas on
                                     (cartas.tipo_id = tipos_cartas.id)
                                     where basic_land = true
                                     )
                  GROUP BY carta_id
                  ORDER BY aparicoes DESC
                  LIMIT n_ranking;
END; $$ LANGUAGE plpgsql;
```