UNIVERSIDADE FEDERAL FLUMINENSE
INSTITUTO DE COMPUTAÇÃO

Projeto de Banco de Dados - 2025/2
Prof. Marcos Bedo

RELATÓRIO DE IMPLEMENTAÇÃO:

SISTEMA DE GERENCIAMENTO DE PLATAFORMAS DE STREAMING

Integrantes:
Gabriel Vieira
Arthur Mota
Danillo Cyrilo
Pedro Piaes
Matheus Andrade

Niterói - RJ
Novembro de 2024

Conteúdo

1 Introdução

Este documento detalha o processo de modelagem e implementação de um banco de dados relacional voltado para o gerenciamento de múltiplas plataformas de streaming, criadores de conteúdo (streamers), usuários e transações financeiras (doações e inscrições). O objetivo principal deste projeto é garantir a integridade referencial dos dados e otimizar a estrutura para um ambiente de alto volume de informações, assegurando que regras de negócio, como tipos de canais e status de pagamentos, sejam rigorosamente seguidas.

2 Definição de Dados (DDL)

A seguir, apresentamos a estrutura física do banco de dados, implementada em SQL (PostgreSQL). O esquema foi organizado no *namespace* `trabalho_bd2` para separar as responsabilidades dentro do banco e evitar conflitos de nomenclatura.

2.1 Tipos de Dados Personalizados e Domínios

Para garantir a consistência dos dados e evitar a inserção de valores inválidos em colunas críticas, optamos pela utilização de tipos enumerados (`ENUM`). Isso restringe a entrada de dados a valores pré-definidos pelas regras de negócio, eliminando a necessidade de validações complexas na camada de aplicação para esses campos.

```
1  CREATE SCHEMA IF NOT EXISTS trabalho_bd2;
2
3  DO $$
4  BEGIN
5      IF NOT EXISTS (SELECT 1 FROM pg_type WHERE typname = 'tipo_canal') THEN
6          CREATE TYPE trabalho_bd2.tipo_canal AS ENUM ('privado', 'público',
       'misto');
7      END IF;
8      IF NOT EXISTS (SELECT 1 FROM pg_type WHERE typname = 'status_doacao')
       THEN
9          CREATE TYPE trabalho_bd2.status_doacao AS ENUM ('recusado', '
       recebido', 'lido');
10     END IF;
11     IF NOT EXISTS (SELECT 1 FROM pg_type WHERE typname = 'nivel_membro')
       THEN
12         CREATE TYPE trabalho_bd2.nivel_membro AS ENUM ('1', '2', '3', '4',
       '5');
13     END IF;
14 END$$;
```

Listing 1: Criação de Schema e Enums

2.2 Entidades Principais

As tabelas a seguir representam as entidades fortes do sistema, que não dependem de outras tabelas para existir. Destacamos a normalização das tabelas `Pais` e

`Conversao` para gerenciar taxas de câmbio de forma centralizada, facilitando atualizações monetárias globais.

```sql
-- Tabela Empresa
CREATE TABLE IF NOT EXISTS trabalho_bd2.Empresa (
    nro INT PRIMARY KEY,
    nome VARCHAR(255) NOT NULL,
    nome_fantasia VARCHAR(255) UNIQUE NOT NULL
);

-- Tabela Conversao (Câmbio)
CREATE TABLE IF NOT EXISTS trabalho_bd2.Conversao (
    moeda VARCHAR(10) PRIMARY KEY,
    nome VARCHAR(100) UNIQUE NOT NULL,
    fator_conver DECIMAL(15,6) NOT NULL
);

-- Tabela Pais
CREATE TABLE IF NOT EXISTS trabalho_bd2.Pais (
    DDI VARCHAR(10) PRIMARY KEY,
    nome VARCHAR(100) UNIQUE NOT NULL,
    moeda VARCHAR(10) NOT NULL,
    FOREIGN KEY (moeda) REFERENCES trabalho_bd2.Conversao(moeda) ON UPDATE CASCADE ON DELETE CASCADE
);

-- Tabela Usuario
CREATE TABLE IF NOT EXISTS trabalho_bd2.Usuario (
    nick VARCHAR(50) PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    data_nasc DATE NOT NULL,
    telefone VARCHAR(20) NOT NULL,
    end_postal VARCHAR(255) NOT NULL,
    pais_residencia VARCHAR(10) NOT NULL,
    FOREIGN KEY (pais_residencia) REFERENCES trabalho_bd2.Pais(DDI) ON UPDATE CASCADE ON DELETE CASCADE
);
```

Listing 2: Entidades Fortes

2.3 Plataformas e Canais

A estrutura central do sistema conecta as empresas às plataformas e os usuários aos seus canais. Utilizamos chaves estrangeiras com a cláusula `ON DELETE CASCADE` para garantir a integridade referencial: se uma plataforma deixar de existir, todos os canais e registros associados a ela serão removidos automaticamente, prevenindo dados órfãos.

```sql
-- Tabela Plataforma
CREATE TABLE IF NOT EXISTS trabalho_bd2.Plataforma (
    nro INT PRIMARY KEY,
    nome VARCHAR(255) UNIQUE NOT NULL,
    qtd_users INT DEFAULT 0,
    empresa_fund INT NOT NULL,
    empresa_respo INT NOT NULL,
    data_fund DATE NOT NULL,
```

```
    FOREIGN KEY (empresa_fund) REFERENCES trabalho_bd2.Empresa(nro) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (empresa_respo) REFERENCES trabalho_bd2.Empresa(nro) ON UPDATE CASCADE ON DELETE CASCADE
);

-- Tabela Canal
CREATE TABLE IF NOT EXISTS trabalho_bd2.Canal (
    nome VARCHAR(255),
    nro_plataforma INT,
    tipo trabalho_bd2.tipo_canal NOT NULL,
    data DATE NOT NULL,
    descricao TEXT,
    qtd_visualizacoes INT DEFAULT 0,
    nick_streamer VARCHAR(50) NOT NULL,
    PRIMARY KEY (nome, nro_plataforma),
    FOREIGN KEY (nro_plataforma) REFERENCES trabalho_bd2.Plataforma(nro) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (nick_streamer) REFERENCES trabalho_bd2.Usuario(nick) ON UPDATE CASCADE ON DELETE CASCADE,
    UNIQUE (nro_plataforma, nick_streamer)
);
```

Listing 3: Estrutura de Plataformas e Canais

2.4 Conteúdo e Interação

Esta seção aborda o armazenamento de conteúdo gerado pelos usuários (vídeos) e as interações (comentários). Um ponto crucial de design foi o uso de chaves primárias compostas na tabela `Video` e `Comentario`, assegurando que um vídeo seja unicamente identificado não apenas pelo seu título, mas também pelo canal e plataforma onde foi publicado.

```
-- Tabela Video
CREATE TABLE IF NOT EXISTS trabalho_bd2.Video (
    nome_canal VARCHAR(255),
    nro_plataforma INT,
    titulo VARCHAR(255),
    dataH TIMESTAMP NOT NULL,
    tema VARCHAR(100) NOT NULL,
    duracao INT NOT NULL,
    visu_simul INT DEFAULT 0,
    visu_total INT DEFAULT 0,
    PRIMARY KEY (nome_canal, nro_plataforma, titulo, dataH),
    FOREIGN KEY (nome_canal, nro_plataforma) REFERENCES trabalho_bd2.Canal(nome, nro_plataforma) ON UPDATE CASCADE ON DELETE CASCADE
);

-- Tabela Comentario
CREATE TABLE IF NOT EXISTS trabalho_bd2.Comentario (
    nome_canal VARCHAR(255),
    nro_plataforma INT,
    titulo_video VARCHAR(255),
    dataH_video TIMESTAMP,
    nick_usuario VARCHAR(50),
    seq INT,
    texto TEXT NOT NULL,
```

```
    dataH TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    coment_on INT,
    PRIMARY KEY (nome_canal, nro_plataforma, titulo_video, dataH_video,
    nick_usuario, seq),
    FOREIGN KEY (nome_canal, nro_plataforma, titulo_video, dataH_video)
    REFERENCES trabalho_bd2.Video(nome_canal, nro_plataforma, titulo, dataH)
     ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (nick_usuario) REFERENCES trabalho_bd2.Usuario(nick) ON
    UPDATE CASCADE ON DELETE CASCADE
);
```

Listing 4: Vídeos e Comentários

2.5 Sistema Financeiro (Herança)

O sistema implementa uma estrutura de generalização/especialização para pagamentos. A tabela `Doacao` serve como entidade genérica contendo os atributos comuns (valor, data, status), enquanto `BitCoin`, `PayPal` e `CartaoCredito` armazenam apenas os atributos específicos de cada método.

```
-- Tabela Genérica de Doação
CREATE TABLE IF NOT EXISTS trabalho_bd2.Doacao (
    nome_canal VARCHAR(255),
    nro_plataforma INT,
    titulo_video VARCHAR(255),
    dataH_video TIMESTAMP,
    nick_usuario VARCHAR(50),
    seq_comentario INT,
    seq_pg INT,
    valor DECIMAL(15,2) NOT NULL,
    status trabalho_bd2.status_doacao NOT NULL,
    data_doacao TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (nome_canal, nro_plataforma, titulo_video, dataH_video,
    nick_usuario, seq_comentario, seq_pg),
    FOREIGN KEY (nome_canal, nro_plataforma, titulo_video, dataH_video,
    nick_usuario, seq_comentario)
        REFERENCES trabalho_bd2.Comentario(nome_canal, nro_plataforma,
    titulo_video, dataH_video, nick_usuario, seq_comentario, seq_pg)
        ON UPDATE CASCADE ON DELETE CASCADE
);

-- Exemplo de Especialização: PayPal
CREATE TABLE IF NOT EXISTS trabalho_bd2.PayPal (
    nome_canal VARCHAR(255),
    nro_plataforma INT,
    titulo_video VARCHAR(255),
    dataH_video TIMESTAMP,
    nick_usuario VARCHAR(50),
    seq_comentario INT,
    seq_doacao INT,
    IdPayPal VARCHAR(255) UNIQUE NOT NULL,
    PRIMARY KEY (nome_canal, nro_plataforma, titulo_video, dataH_video,
    nick_usuario, seq_comentario, seq_doacao),
    FOREIGN KEY (nome_canal, nro_plataforma, titulo_video, dataH_video,
    nick_usuario, seq_comentario, seq_doacao)
        REFERENCES trabalho_bd2.Doacao(nome_canal, nro_plataforma,
    titulo_video, dataH_video, nick_usuario, seq_comentario, seq_pg)
```

```
        ON UPDATE CASCADE ON DELETE CASCADE
);
```

Listing 5: Sistema de Doações e Pagamentos

3 Consultas e Procedimentos Armazenados

Nesta seção, apresentamos as soluções para as consultas requisitadas. Implementamos as consultas utilizando funções em PL/pgSQL com retorno de tabela (`RETURNS TABLE`), permitindo a inclusão de parâmetros opcionais para filtragem dinâmica.

3.1 Relatórios de Patrocínio e Inscritos

1. Identificar quais são os canais patrocinados e os valores de patrocínio pagos por empresa. Dar a opção de filtrar os resultados por empresa como um parâmetro opcional na forma de uma stored procedure.

```
CREATE OR REPLACE FUNCTION listar_patrocinios(empresa_filter INT DEFAULT
    NULL)
RETURNS TABLE (
    nome_empresa VARCHAR(255),
    nome_canal VARCHAR(255),
    nome_plataforma VARCHAR(255),
    valor_patrocinio DECIMAL(15,2)
) AS $$
BEGIN
    RETURN QUERY
    SELECT e.nome, pat.nome_canal, pl.nome, pat.valor
    FROM Patrocinio pat
    JOIN Empresa e ON pat.nro_empresa = e.nro
    JOIN Plataforma pl ON pat.nro_plataforma = pl.nro
    WHERE (empresa_filter IS NULL OR pat.nro_empresa = empresa_filter)
    ORDER BY e.nome, pat.valor DESC;
END;
$$ LANGUAGE plpgsql;
```

Listing 6: Consulta 1: Canais Patrocinados

2. Descobrir de quantos canais cada usuário é membro e a soma do valor desembolsado por usuário por mês. Dar a opção de filtrar os resultados por usuário como um parâmetro opcional na forma de uma stored procedure.

```
CREATE OR REPLACE FUNCTION membros_inscricoes(usuario_filter VARCHAR(50)
    DEFAULT NULL)
RETURNS TABLE (
    nick_usuario VARCHAR(50),
    qtd_canais BIGINT,
    valor_mensal_total DECIMAL(10,2)
) AS $$
BEGIN
    RETURN QUERY
    SELECT i.nick_membro,
           COUNT(*) as qtd_canais,
```

```
            SUM(nc.valor) as valor_mensal_total
    FROM Inscricao i
    JOIN NivelCanal nc ON i.nome_canal = nc.nome_canal
                      AND i.nro_plataforma = nc.nro_plataforma
                      AND i.nivel = nc.nivel
    WHERE (usuario_filter IS NULL OR i.nick_membro = usuario_filter)
    GROUP BY i.nick_membro
    ORDER BY valor_mensal_total DESC;
END;
$$ LANGUAGE plpgsql;
```

Listing 7: Consulta 2: Membros e Valor por Usuário

3.2 Relatórios de Doações

3. Listar e ordenar os canais que já receberam doações e a soma dos valores recebidos em doação. Dar a opção de filtrar os resultados por canal como um parâmetro opcional na forma de uma stored procedure.

```
CREATE OR REPLACE FUNCTION doacoes_por_canal(canal_filter VARCHAR(255)
    DEFAULT NULL)
RETURNS TABLE (
    nome_canal VARCHAR(255),
    nome_plataforma VARCHAR(255),
    total_doacoes DECIMAL(15,2)
) AS $$
BEGIN
    RETURN QUERY
    SELECT c.nome, pl.nome, SUM(d.valor) as total_doacoes
    FROM Doacao d
    JOIN Canal c ON d.nome_canal = c.nome AND d.nro_plataforma = c.
    nro_plataforma
    JOIN Plataforma pl ON c.nro_plataforma = pl.nro
    WHERE (canal_filter IS NULL OR c.nome = canal_filter)
      AND d.status != 'recusado' -- Correção: Ignora doações recusadas
    GROUP BY c.nome, pl.nome
    ORDER BY total_doacoes DESC;
END;
$$ LANGUAGE plpgsql;
```

Listing 8: Consulta 3: Doações Recebidas por Canal (Exclui Recusadas)

4. Listar a soma das doações geradas pelos comentários que foram lidos por vídeo. Dar a opção de filtrar os resultados por vídeo como um parâmetro opcional na forma de uma stored procedure.

```
CREATE OR REPLACE FUNCTION doacoes_lidas_por_video(
    video_titulo_filter VARCHAR(255) DEFAULT NULL,
    video_data_filter TIMESTAMP DEFAULT NULL
) RETURNS TABLE (
    titulo_video VARCHAR(255),
    dataH_video TIMESTAMP,
    total_doacoes_lidas DECIMAL(15,2)
) AS $$
BEGIN
    RETURN QUERY
```

```
    SELECT v.titulo, v.dataH, COALESCE(SUM(d.valor), 0.00) as
    total_doacoes_lidas
    FROM Video v
    LEFT JOIN Doacao d ON d.nome_canal = v.nome_canal
                       AND d.nro_plataforma = v.nro_plataforma
                       AND d.titulo_video = v.titulo
                       AND d.dataH_video = v.dataH
                       AND d.status = 'lido' -- Status 'lido' já exclui '
    recusado' implicitamente
    WHERE (video_titulo_filter IS NULL OR v.titulo = video_titulo_filter)
      AND (video_data_filter IS NULL OR v.dataH = video_data_filter)
    GROUP BY v.titulo, v.dataH
    HAVING COALESCE(SUM(d.valor), 0.00) > 0
    ORDER BY total_doacoes_lidas DESC;
END;
$$ LANGUAGE plpgsql;
```

Listing 9: Consulta 4: Doações em Comentários Lidos

3.3 Rankings (Top K)

5. Listar e ordenar os k canais que mais recebem patrocínio e os valores recebidos.

```
CREATE OR REPLACE FUNCTION top_canais_patrocinio(k INT)
RETURNS TABLE (
    nome_canal VARCHAR(255),
    nome_plataforma VARCHAR(255),
    total_patrocinio DECIMAL(15,2)
) AS $$
BEGIN
    RETURN QUERY
    SELECT c.nome, pl.nome, COALESCE(SUM(pat.valor), 0.00) as
    total_patrocinio
    FROM Canal c
    JOIN Plataforma pl ON c.nro_plataforma = pl.nro
    LEFT JOIN Patrocinio pat ON c.nome = pat.nome_canal AND c.
    nro_plataforma = pat.nro_plataforma
    GROUP BY c.nome, pl.nome
    ORDER BY total_patrocinio DESC
    LIMIT k;
END;
$$ LANGUAGE plpgsql;
```

Listing 10: Consulta 5: Top Patrocínios

6. Listar e ordenar os k canais que mais recebem aportes de membros e os valores recebidos.

```
CREATE OR REPLACE FUNCTION top_canais_membros(k INT)
RETURNS TABLE (
    nome_canal VARCHAR(255),
    nome_plataforma VARCHAR(255),
    total_membros DECIMAL(10,2)
) AS $$
BEGIN
    RETURN QUERY
```

```
    SELECT c.nome, pl.nome, COALESCE(SUM(nc.valor), 0.00) as total_membros
    FROM Canal c
    JOIN Plataforma pl ON c.nro_plataforma = pl.nro
    LEFT JOIN Inscricao i ON c.nome = i.nome_canal AND c.nro_plataforma = i
    .nro_plataforma
    LEFT JOIN NivelCanal nc ON i.nome_canal = nc.nome_canal
                           AND i.nro_plataforma = nc.nro_plataforma
                           AND i.nivel = nc.nivel
    GROUP BY c.nome, pl.nome
    ORDER BY total_membros DESC
    LIMIT k;
END;
$$ LANGUAGE plpgsql;
```

Listing 11: Consulta 6: Top Membros

7. Listar e ordenar os k canais que mais receberam doações considerando todos os vídeos.

```
CREATE OR REPLACE FUNCTION top_canais_doacoes(k INT)
RETURNS TABLE (
    nome_canal VARCHAR(255),
    nome_plataforma VARCHAR(255),
    total_doacoes DECIMAL(15,2)
) AS $$
BEGIN
    RETURN QUERY
    SELECT c.nome, pl.nome, COALESCE(SUM(d.valor), 0.00) as total_doacoes
    FROM Canal c
    JOIN Plataforma pl ON c.nro_plataforma = pl.nro
    LEFT JOIN Doacao d ON c.nome = d.nome_canal
                      AND c.nro_plataforma = d.nro_plataforma
                      AND d.status != 'recusado' -- Correção: Filtro
    adicionado
    GROUP BY c.nome, pl.nome
    ORDER BY total_doacoes DESC
    LIMIT k;
END;
$$ LANGUAGE plpgsql;
```

Listing 12: Consulta 7: Top Doações (Exclui Recusadas)

3.4 Faturamento Consolidado

8. Listar os k canais que mais faturam considerando as três fontes de receita: patrocínio, membros inscritos e doações.

```
CREATE OR REPLACE FUNCTION top_canais_faturamento(k INT)
RETURNS TABLE (
    nome_canal VARCHAR(255),
    nome_plataforma VARCHAR(255),
    total_patrocinio DECIMAL(15,2),
    total_membros DECIMAL(10,2),
    total_doacoes DECIMAL(15,2),
    faturamento_total DECIMAL(15,2)
) AS $$
BEGIN
```

```
    RETURN QUERY
    SELECT
        v.nome_canal,
        p.nome AS nome_plataforma,
        v.receita_patrocinio AS total_patrocinio,
        v.receita_membros AS total_membros,
        v.receita_doacoes AS total_doacoes,
        v.receita_total AS faturamento_total
    FROM vw_receita_total_canal v
    JOIN Plataforma p ON v.nro_plataforma = p.nro
    ORDER BY v.receita_total DESC
    LIMIT k;
END;
$$ LANGUAGE plpgsql;
```

Listing 13: Consulta 8: Faturamento Total Consolidado

4 Otimização e Vistas (Views)

Esta seção aborda as estratégias de otimização de performance e simplificação de consultas complexas através do uso de Views, Views Materializadas e Índices específicos para cargas de trabalho de leitura intensiva.

4.1 Views e Materialized Views

View de Patrocínios em Canais (`vw_patrocinios_canais`)
Objetivo: Listar canais patrocinados, valor do patrocínio e empresa patrocinadora de forma desnormalizada para relatórios rápidos.

```
CREATE OR REPLACE VIEW vw_patrocinios_canais AS
SELECT p.nro_empresa,
       e.nome AS empresa_nome,
       p.nome_canal,
       p.nro_plataforma,
       p.valor,
       p.data_inicio
FROM Patrocinio p
JOIN Empresa e ON p.nro_empresa = e.nro;
```

View de Membros Inscritos (`vw_membros_inscritos`)
Objetivo: Calcular, para cada usuário, de quantos canais ele é membro e a soma mensal desembolsada. Devido à alta cardinalidade (milhões de usuários) e ao padrão de acesso pontual (consulta por ID), optou-se por uma **View Virtual** em vez de materializada, evitando custos proibitivos de armazenamento e refresh. A performance é garantida pelo índice criado na tabela `Inscricao`.

```
CREATE OR REPLACE VIEW vw_membros_inscritos AS
SELECT i.nick_membro,
       COUNT(*) AS qtd_canais,
       SUM(n.valor) AS valor_mensal_total
FROM Inscricao i
JOIN NivelCanal n ON (i.nome_canal = n.nome_canal
                  AND i.nro_plataforma = n.nro_plataforma
                  AND i.nivel = n.nivel)
GROUP BY i.nick_membro;
```

View de Doações por Canal (Materializada - `vw_doacoes_canais`)
Objetivo: Pré-calcular a soma total de doações por canal, evitando varreduras completas na tabela de doações para dashboards.

```sql
CREATE MATERIALIZED VIEW vw_doacoes_canais AS
SELECT d.nome_canal,
       d.nro_plataforma,
       SUM(d.valor) AS total_doacoes,
       COUNT(*) AS qtd_doacoes
FROM Doacao d
WHERE d.status != 'recusado'
GROUP BY d.nome_canal, d.nro_plataforma;
```

View de Doações Lidas por Vídeo (`vw_doacoes_lidas_por_video`)
Objetivo: Filtrar e somar apenas as doações que foram efetivamente lidas (status 'lido') por vídeo.

```sql
CREATE OR REPLACE VIEW vw_doacoes_lidas_por_video AS
SELECT d.nome_canal,
       d.nro_plataforma,
       d.titulo_video,
       d.dataH_video,
       SUM(d.valor) FILTER (WHERE d.status = 'lido') AS total_doacoes_lidas
FROM Doacao d
GROUP BY d.nome_canal, d.nro_plataforma, d.titulo_video, d.dataH_video;
```

View de Receita Total por Canal (Materializada - `vw_receita_total_canal`)
Objetivo: Consolidar todas as fontes de receita (patrocínio, membros e doações) em uma única visão unificada. Devido ao alto custo computacional de agregar três fontes distintas e realizar junções completas (FULL OUTER JOIN), esta visão é candidata ideal para materialização, otimizando drasticamente o relatório de ranking financeiro (Consulta 8).

```sql
CREATE MATERIALIZED VIEW vw_receita_total_canal AS
WITH pat AS (
    SELECT nome_canal, nro_plataforma, SUM(valor) AS receita_patrocinio
    FROM Patrocinio
    GROUP BY nome_canal, nro_plataforma
),
memb AS (
    SELECT i.nome_canal, i.nro_plataforma, SUM(n.valor) AS receita_membros
    FROM Inscricao i
    JOIN NivelCanal n ON i.nome_canal = n.nome_canal
                     AND i.nro_plataforma = n.nro_plataforma
                     AND i.nivel = n.nivel
    GROUP BY i.nome_canal, i.nro_plataforma
),
doa AS (
    SELECT nome_canal, nro_plataforma, SUM(valor) AS receita_doacoes
    FROM Doacao
    WHERE status != 'recusado'
    GROUP BY nome_canal, nro_plataforma
)
SELECT COALESCE(pat.nome_canal, memb.nome_canal, doa.nome_canal) AS
    nome_canal,
       COALESCE(pat.nro_plataforma, memb.nro_plataforma, doa.nro_plataforma
    ) AS nro_plataforma,
       COALESCE(pat.receita_patrocinio, 0) AS receita_patrocinio,
       COALESCE(memb.receita_membros, 0) AS receita_membros,
```

```
25        COALESCE(doa.receita_doacoes, 0) AS receita_doacoes,
26        COALESCE(pat.receita_patrocinio, 0) +
27        COALESCE(memb.receita_membros, 0) +
28        COALESCE(doa.receita_doacoes, 0) AS receita_total
29 FROM pat
30 FULL OUTER JOIN memb USING (nome_canal, nro_plataforma)
31 FULL OUTER JOIN doa USING (nome_canal, nro_plataforma);
```

4.2 Índices de Performance

Abaixo estão os índices criados para otimizar as consultas mais críticas do sistema.

idx_inscricao_membro_btree

Tipo: B-TREE

Justificativa: Essencial para dar suporte à `vw_membros_inscritos`, permitindo que a busca por estatísticas de um usuário específico seja realizada em tempo logarítmico, sem varrer toda a tabela de inscrições.

```
1 CREATE INDEX idx_inscricao_membro_btree ON Inscricao (nick_membro) USING
    BTREE;
```

idx_patrocinio_valor_btree

Tipo: B-TREE

Justificativa: Otimiza ordenações e consultas por faixa de valor, essenciais para relatórios de "Maiores Patrocínios".

```
1 CREATE INDEX idx_patrocinio_valor_btree ON Patrocinio (valor) USING BTREE;
```

idx_patrocinio_empresa_btree

Tipo: B-TREE

Justificativa: Acelera junções e filtros por empresa patrocinadora (`WHERE nro_empresa = ...`), muito usado na Consulta 1.

```
1 CREATE INDEX idx_patrocinio_empresa_btree ON Patrocinio (nro_empresa) USING
    BTREE;
```

idx_doacao_status_hash

Tipo: HASH

Justificativa: Consultas financeiras filtram frequentemente por igualdade de status (ex: `status = 'lido'`). Índices Hash são O(1) para igualdade.

```
1 CREATE INDEX idx_doacao_status_hash ON Doacao (status) USING HASH;
```

idx_doacao_canal_btree

Tipo: B-TREE

Justificativa: Facilita o agrupamento e a soma de doações por canal, suportando as agregações da `vw_doacoes_canais`.

```
1 CREATE INDEX idx_doacao_canal_btree ON Doacao(nome_canal, nro_plataforma)
    USING BTREE;
```

idx_nivelcanal_lookup_btree

Tipo: B-TREE

Justificativa: Permite lookup rápido do valor do nível ao calcular a receita de membros, evitando full table scans na tabela `NivelCanal`.

```
1 CREATE INDEX idx_nivelcanal_lookup_btree ON NivelCanal (nome_canal,
    nro_plataforma, nivel) USING BTREE;
```

5 Triggers e Regras Ativas (Consistência)

Esta seção apresenta os gatilhos (triggers) desenvolvidos com foco na manutenção ativa da consistência dos dados. Diferente de validações passivas (que apenas rejeitam dados), estes triggers agem automaticamente para atualizar atributos derivados, realizar conversões ou manter logs de auditoria, garantindo que o banco de dados permaneça sincronizado e confiável.

5.1 Atualização Automática de Usuários na Plataforma

Este trigger mantém o contador de usuários na tabela Plataforma sempre atualizado. Ao inserir ou remover um usuário da tabela de relação PlataformaUsuario, ele incrementa ou decrementa automaticamente o campo `qtd_users`, otimizando consultas futuras.

```
CREATE OR REPLACE FUNCTION atualizar_qtd_users()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        UPDATE trabalho_bd2.Plataforma
        SET qtd_users = qtd_users + 1
        WHERE nro = NEW.nro_plataforma;
        RETURN NEW;
    ELSIF TG_OP = 'DELETE' THEN
        UPDATE trabalho_bd2.Plataforma
        SET qtd_users = qtd_users - 1
        WHERE nro = OLD.nro_plataforma;
        RETURN OLD;
    END IF;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_atualizar_qtd_users
AFTER INSERT OR DELETE ON trabalho_bd2.PlataformaUsuario
FOR EACH ROW EXECUTE FUNCTION atualizar_qtd_users();
```

5.2 Atualização Automática de Visualizações no Canal

Mantém a soma total de visualizações de um canal atualizada em tempo real. Quando um vídeo sofre alterações em suas visualizações, o total do canal é ajustado automaticamente.

```
CREATE OR REPLACE FUNCTION atualizar_visualizacoes_canal()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        UPDATE trabalho_bd2.Canal
        SET qtd_visualizacoes = qtd_visualizacoes + NEW.visu_total
        WHERE nome = NEW.nome_canal AND nro_plataforma = NEW.nro_plataforma
    ;
        RETURN NEW;
    ELSIF TG_OP = 'UPDATE' THEN
        UPDATE trabalho_bd2.Canal
```

```
        SET qtd_visualizacoes = qtd_visualizacoes - OLD.visu_total + NEW.visu_total
        WHERE nome = NEW.nome_canal AND nro_plataforma = NEW.nro_plataforma;
        RETURN NEW;
    ELSIF TG_OP = 'DELETE' THEN
        UPDATE trabalho_bd2.Canal
        SET qtd_visualizacoes = qtd_visualizacoes - OLD.visu_total
        WHERE nome = OLD.nome_canal AND nro_plataforma = OLD.nro_plataforma;
        RETURN OLD;
    END IF;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_atualizar_visualizacoes_canal
AFTER INSERT OR UPDATE OR DELETE ON trabalho_bd2.Video
FOR EACH ROW EXECUTE FUNCTION atualizar_visualizacoes_canal();
```

5.3 Atualizar Status de Doação ao Remover Comentário

Define um comportamento de ”fail-safe”: se um comentário com doação for excluído, a doação não fica órfã nem é excluída, mas sim marcada como ’recusada’ para manter o histórico contábil.

```
CREATE OR REPLACE FUNCTION atualizar_status_doacao_comentario()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE trabalho_bd2.Doacao
    SET status = 'recusado'
    WHERE nome_canal = OLD.nome_canal
    AND nro_plataforma = OLD.nro_plataforma
    AND titulo_video = OLD.titulo_video
    AND dataH_video = OLD.dataH_video
    AND nick_usuario = OLD.nick_usuario
    AND seq_comentario = OLD.seq;

    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_atualizar_status_doacao_comentario
AFTER DELETE ON trabalho_bd2.Comentario
FOR EACH ROW EXECUTE FUNCTION atualizar_status_doacao_comentario();
```

5.4 Gestão de Arrecadação por Vídeo

Mantém um atributo `valor_arrecadado` na tabela Video. Este trigger é inteligente: ele soma valores quando uma doação entra, mas subtrai automaticamente se o status da doação for alterado para ’recusado’ posteriormente.

```
-- Nota: Assume a existencia da coluna valor_arrecadado na tabela Video
CREATE OR REPLACE FUNCTION atualizar_arrecadacao_video()
RETURNS TRIGGER AS $$
```

```
BEGIN
    -- Caso 1: Nova doacao (assume status inicial valido)
    IF TG_OP = 'INSERT' THEN
        UPDATE trabalho_bd2.Video
        SET valor_arrecadado = COALESCE(valor_arrecadado, 0) + NEW.valor
        WHERE nome_canal = NEW.nome_canal AND titulo = NEW.titulo_video;
        RETURN NEW;

    -- Caso 2: Atualizacao de status (se virar recusado, estorna o valor)
    ELSIF TG_OP = 'UPDATE' THEN
        IF OLD.status != 'recusado' AND NEW.status = 'recusado' THEN
            UPDATE trabalho_bd2.Video
            SET valor_arrecadado = valor_arrecadado - OLD.valor
            WHERE nome_canal = OLD.nome_canal AND titulo = OLD.titulo_video
    ;
        END IF;
        RETURN NEW;
    END IF;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_atualizar_arrecadacao
AFTER INSERT OR UPDATE ON trabalho_bd2.Doacao
FOR EACH ROW EXECUTE FUNCTION atualizar_arrecadacao_video();
```

5.5 Padronização Monetária Automática

Ao inserir uma doação, este trigger consulta a tabela de conversão e a moeda do país do usuário para calcular e armazenar automaticamente o valor padronizado em Dólar, facilitando relatórios financeiros globais.

```
CREATE OR REPLACE FUNCTION converter_moeda_doacao()
RETURNS TRIGGER AS $$
DECLARE
    moeda_origem VARCHAR(10);
    taxa DECIMAL(15,6);
BEGIN
    -- Busca a moeda do pais do usuario
    SELECT p.moeda INTO moeda_origem
    FROM trabalho_bd2.Usuario u
    JOIN trabalho_bd2.Pais p ON u.pais_residencia = p.DDI
    WHERE u.nick = NEW.nick_usuario;

    -- Busca a taxa de conversao para dolar (exemplo base)
    SELECT fator_conver INTO taxa
    FROM trabalho_bd2.Conversao
    WHERE moeda = moeda_origem;

    -- Assume coluna valor_convertido na tabela Doacao
    -- NEW.valor_convertido := NEW.valor * taxa;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_converter_moeda
```

```
26 BEFORE INSERT ON trabalho_bd2.Doacao
27 FOR EACH ROW EXECUTE FUNCTION converter_moeda_doacao();
```

5.6 Auditoria de Mudança de Nick

Cria um histórico de segurança. Quando um usuário altera seu Nick, o sistema grava o identificador antigo em uma tabela de log, permitindo rastrear a identidade do usuário ao longo do tempo. (Nota: A atualização nas tabelas relacionadas é feita automaticamente pelo `ON UPDATE CASCADE`).

```
1  CREATE OR REPLACE FUNCTION auditar_mudanca_nick()
2  RETURNS TRIGGER AS $$
3  BEGIN
4      IF OLD.nick != NEW.nick THEN
5          -- Assume tabela de log criada para este fim
6          INSERT INTO trabalho_bd2.Log_Historico_Nick (nick_antigo, nick_novo
       , data_alteracao)
7          VALUES (OLD.nick, NEW.nick, CURRENT_TIMESTAMP);
8      END IF;
9      RETURN NEW;
10 END;
11 $$ LANGUAGE plpgsql;
12
13 CREATE TRIGGER trg_auditar_nick
14 BEFORE UPDATE ON trabalho_bd2.Usuario
15 FOR EACH ROW EXECUTE FUNCTION auditar_mudanca_nick();
```

5.7 Sincronia Geográfica de Streamer

Garante que, se um streamer mudar seu país de residência no cadastro principal, a tabela específica `StreamerPais` (que gerencia questões legais/fiscais) seja atualizada automaticamente para refletir o novo domicílio.

```
1  CREATE OR REPLACE FUNCTION sincronizar_pais_streamer()
2  RETURNS TRIGGER AS $$
3  BEGIN
4      IF OLD.pais_residencia != NEW.pais_residencia THEN
5          UPDATE trabalho_bd2.StreamerPais
6          SET ddi_pais = NEW.pais_residencia
7          WHERE nick_streamer = NEW.nick;
8      END IF;
9      RETURN NEW;
10 END;
11 $$ LANGUAGE plpgsql;
12
13 CREATE TRIGGER trg_sincronizar_pais
14 AFTER UPDATE ON trabalho_bd2.Usuario
15 FOR EACH ROW EXECUTE FUNCTION sincronizar_pais_streamer();
```

5.8 Gestão de Inscrições em Canais Privados

Implementa uma regra de negócio restritiva: se um canal alterar seu status para 'privado' (ou for suspenso), todas as inscrições ativas são automaticamente can-

celadas ou suspensas, mantendo a coerência de acesso ao conteúdo.

```
CREATE OR REPLACE FUNCTION gerir_canal_privado()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.tipo = 'privado' AND OLD.tipo != 'privado' THEN
        -- Exemplo: Remove inscricoes pois canal privado nao permite acesso publico
        DELETE FROM trabalho_bd2.Inscricao
        WHERE nome_canal = NEW.nome AND nro_plataforma = NEW.nro_plataforma;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_gerir_privado
AFTER UPDATE ON trabalho_bd2.Canal
FOR EACH ROW EXECUTE FUNCTION gerir_canal_privado();
```

6 Conclusão

O desenvolvimento deste projeto permitiu a aplicação prática dos conceitos de modelagem de dados, implementação de consultas complexas e regras ativas. O esquema de banco de dados desenvolvido atende robustamente às necessidades de gerenciamento de plataformas de streaming, garantindo a integridade dos dados através de chaves estrangeiras, restrições e triggers. As consultas armazenadas fornecem uma interface flexível para extração de relatórios gerenciais, cobrindo aspectos de patrocínio, engajamento de usuários e análise financeira consolidada.