**Centro Universitário UNISATC**

Engenharia de Software 3a fase – Banco de Dados II – Prof.Cristiane Pavei Fernandes

**TRABALHO FINAL COM BASE EM METODOLOGIAS ATIVAS DE APRENDIZAGEM**

**Projeto de banco de dados para um sistema de SpotMusic**

Gustavo de Freitas Cardoso

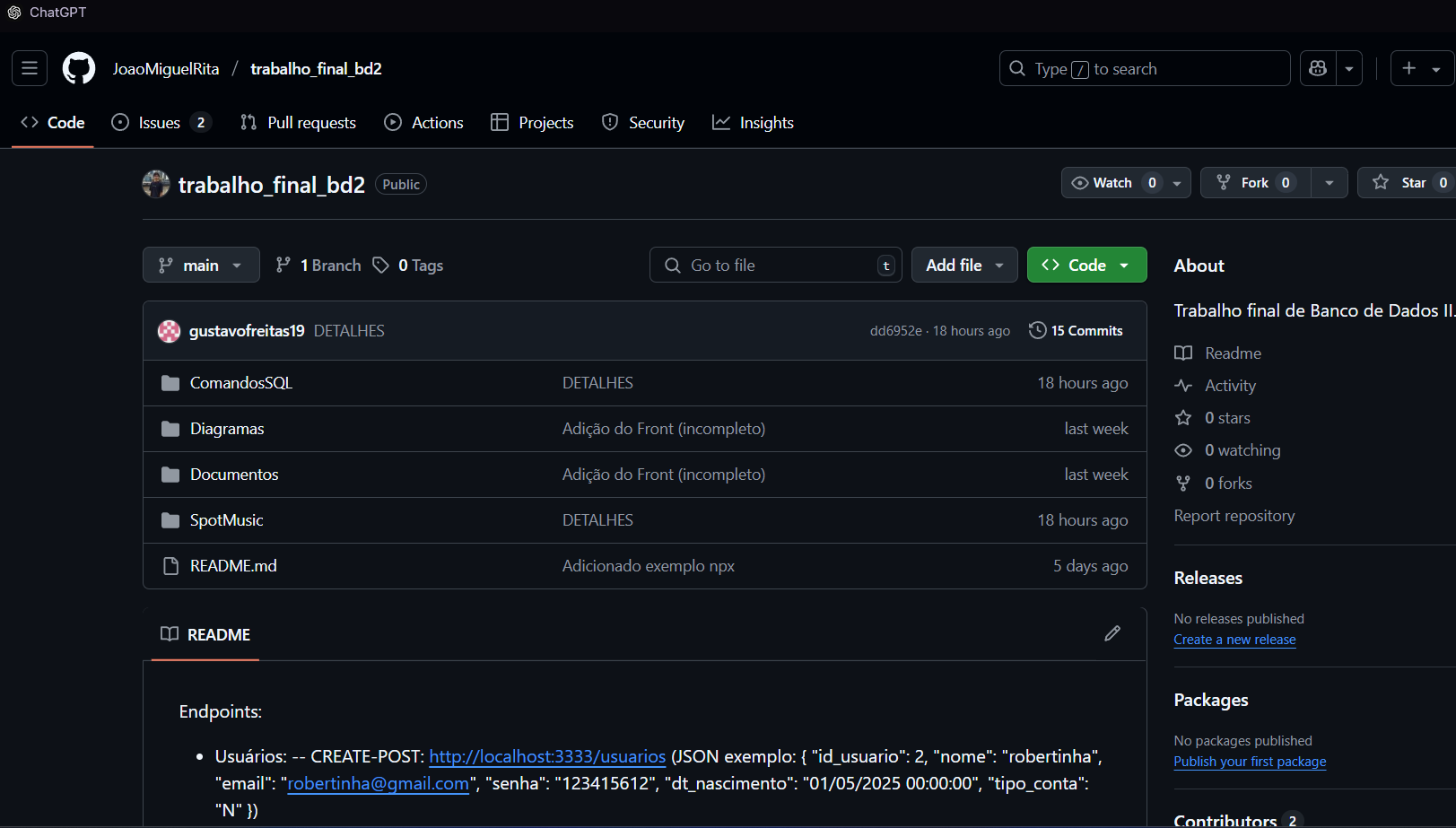
João Miguel Fortunato Rita

Conta GitHub/Google Drive (compartilhada)

Criciúma, 10/06/2025

**URL do projeto no GitHub**

<https://github.com/JoaoMiguelRita/trabalho_final_bd2>

****

**Modelo ER Físico**

**Diagrama, Esquemático

O conteúdo gerado por IA pode estar incorreto.**

**Dicionário de Dados**

**Interface gráfica do usuário, Calendário

O conteúdo gerado por IA pode estar incorreto.**

**Interface gráfica do usuário, Tabela

O conteúdo gerado por IA pode estar incorreto.**

**Tela de computador com texto preto sobre fundo branco

O conteúdo gerado por IA pode estar incorreto.**

**Interface gráfica do usuário, Texto, Aplicativo

O conteúdo gerado por IA pode estar incorreto.**

**Uma imagem contendo Interface gráfica do usuário

O conteúdo gerado por IA pode estar incorreto.**

**Tela de computador com texto preto sobre fundo branco

O conteúdo gerado por IA pode estar incorreto.**

**Tela de computador com texto preto sobre fundo branco

O conteúdo gerado por IA pode estar incorreto.**

**Script dos comandos DDL para criação do Banco de dados**

CREATE TABLE usuario (

    id\_usuario INT PRIMARY KEY,

    nome VARCHAR(100) NOT NULL,

    email VARCHAR(100) UNIQUE NOT NULL,

    senha VARCHAR(100) NOT NULL,

    dt\_nascimento DATE NOT NULL,

    tipo\_conta char (1)

);

CREATE TABLE artista (

    id\_artista INT PRIMARY KEY,

    nome VARCHAR(100) NOT NULL,

    nacionalidade VARCHAR(50)

);

CREATE TABLE album (

    id\_album INT PRIMARY KEY,

    titulo VARCHAR(100) NOT NULL,

    ano\_lancamento INT,

    id\_artista INT,

    FOREIGN KEY (id\_artista) REFERENCES Artista(id\_artista)

);

CREATE TABLE musica (

    id\_musica INT PRIMARY KEY,

    titulo VARCHAR(100) NOT NULL,

    genero VARCHAR(50),

    id\_album INT,

    id\_artista INT,

    FOREIGN KEY (id\_album) REFERENCES Album(id\_album),

    FOREIGN KEY (id\_artista) REFERENCES Artista(id\_artista)

);

CREATE TABLE playlist (

    id\_playlist INT PRIMARY KEY,

    nome VARCHAR(100) NOT NULL,

    id\_usuario INT,

    FOREIGN KEY (id\_usuario) REFERENCES Usuario(id\_usuario)

);

CREATE TABLE playlist\_musica (

    id\_playlist INT,

    id\_musica INT,

    ordem INT,

    PRIMARY KEY (id\_playlist, id\_musica),

    FOREIGN KEY (id\_playlist) REFERENCES Playlist(id\_playlist),

    FOREIGN KEY (id\_musica) REFERENCES Musica(id\_musica)

);

CREATE TABLE historico\_reproducao (

    id\_usuario INT,

    id\_musica INT,

    data\_hora TIMESTAMP,

    PRIMARY KEY (id\_usuario, id\_musica, data\_hora),

    FOREIGN KEY (id\_usuario) REFERENCES Usuario(id\_usuario),

    FOREIGN KEY (id\_musica) REFERENCES Musica(id\_musica)

);

CREATE TABLE curtidas (

    id\_usuario INT,

    id\_musica INT,

    data\_curtida TIMESTAMP,

    PRIMARY KEY (id\_usuario, id\_musica),

    FOREIGN KEY (id\_usuario) REFERENCES Usuario(id\_usuario),

    FOREIGN KEY (id\_musica) REFERENCES Musica(id\_musica)

);

**Script que popula as tabelas do Banco de dados**

-- usuario

INSERT INTO usuario

       ("nome", "email", "senha", "dt\_nascimento", "tipo\_conta", "createdAt", "updatedAt")

VALUES

('Alice Costa', 'alice@example.com', 'senha123', '1990-04-12', 'normal', NOW(), NOW()),

('Bruno Lima', 'bruno@example.com', 'senha123', '1985-06-23', 'premium', NOW(), NOW()),

('Carlos Dias', 'carlos@example.com', 'senha123', '2000-10-08', 'administrador', NOW(), NOW()),

('Daniela Rocha', 'daniela@example.com', 'senha123', '1995-02-17', 'normal', NOW(), NOW()),

('Eduardo Silva', 'eduardo@example.com', 'senha123', '1988-12-01', 'premium', NOW(), NOW()),

('Fernanda Luz', 'fernanda@example.com', 'senha123', '1992-07-09', 'normal', NOW(), NOW()),

('Gustavo Torres', 'gustavo@example.com', 'senha123', '1999-03-03', 'administrador', NOW(), NOW()),

('Helena Almeida', 'helena@example.com', 'senha123', '1997-08-20', 'premium', NOW(), NOW()),

('Igor Mendes', 'igor@example.com', 'senha123', '1994-11-15', 'normal', NOW(), NOW()),

('Juliana Farias', 'juliana@example.com', 'senha123', '2001-01-30', 'premium', NOW(), NOW());

-- artista

INSERT INTO artista

       ("nome", "nacionalidade", "createdAt", "updatedAt")

VALUES

('Banda Alpha', 'Brasil', NOW(), NOW()),

('DJ Max', 'EUA', NOW(), NOW()),

('Luna Melo', 'Brasil', NOW(), NOW()),

('The Sunsetters', 'Inglaterra', NOW(), NOW()),

('Sophie Blue', 'Canadá', NOW(), NOW()),

('Grupo Raiz', 'Brasil', NOW(), NOW()),

('Eletronic Pulse', 'Alemanha', NOW(), NOW()),

('João Pires', 'Portugal', NOW(), NOW()),

('The Rockets', 'EUA', NOW(), NOW()),

('Marina Sky', 'Austrália', NOW(), NOW());

-- album

INSERT INTO album

       ("titulo", "ano\_lancamento", "id\_artista", "createdAt", "updatedAt")

VALUES

('Aurora Boreal', 2020, 1, NOW(), NOW()),

('Batidas Urbanas', 2018, 2, NOW(), NOW()),

('Caminhos', 2021, 3, NOW(), NOW()),

('Golden Times', 2015, 4, NOW(), NOW()),

('Blue Horizon', 2022, 5, NOW(), NOW()),

('Raízes', 2019, 6, NOW(), NOW()),

('Frequência Total', 2020, 7, NOW(), NOW()),

('Canções do Sul', 2021, 8, NOW(), NOW()),

('Rockets Reloaded', 2017, 9, NOW(), NOW()),

('Sky High', 2023, 10, NOW(), NOW());

-- musica

INSERT INTO musica

       ("titulo", "genero", "id\_album", "id\_artista", "createdAt", "updatedAt")

VALUES

('Nas Estrelas', 'Pop', 1, 1, NOW(), NOW()),

('Ritmo da Rua', 'Eletrônica', 2, 2, NOW(), NOW()),

('Horizonte Aberto', 'MPB', 3, 3, NOW(), NOW()),

('Golden Days', 'Rock', 4, 4, NOW(), NOW()),

('Ocean Deep', 'Indie', 5, 5, NOW(), NOW()),

('Meu Sertão', 'Sertanejo', 6, 6, NOW(), NOW()),

('Sintonia', 'Eletrônica', 7, 7, NOW(), NOW()),

('Fado Novo', 'Fado', 8, 8, NOW(), NOW()),

('Rocket Fuel', 'Rock', 9, 9, NOW(), NOW()),

('Voar Mais Alto', 'Pop', 10, 10, NOW(), NOW());

-- playlist

INSERT INTO playlist

       ("nome", "id\_usuario", "createdAt", "updatedAt")

VALUES

('Treino Pesado', 1, NOW(), NOW()),

('Relaxamento', 2, NOW(), NOW()),

('Hits 2020', 3, NOW(), NOW()),

('Favoritas da Dani', 4, NOW(), NOW()),

('MPB Top', 5, NOW(), NOW()),

('Eletrônicas', 6, NOW(), NOW()),

('Rock Clássico', 7, NOW(), NOW()),

('Só Sertanejo', 8, NOW(), NOW()),

('Vibe Indie', 9, NOW(), NOW()),

('Músicas de Viagem', 10, NOW(), NOW());

-- playlist\_musica

INSERT INTO playlist\_musica

       ("id\_playlist", "id\_musica", "ordem", "createdAt", "updatedAt")

VALUES

(1, 2, 1, NOW(), NOW()),

(1, 7, 2, NOW(), NOW()),

(2, 5, 1, NOW(), NOW()),

(2, 10, 2, NOW(), NOW()),

(3, 1, 1, NOW(), NOW()),

(3, 2, 2, NOW(), NOW()),

(4, 3, 1, NOW(), NOW()),

(4, 6, 2, NOW(), NOW()),

(5, 3, 1, NOW(), NOW()),

(6, 7, 1, NOW(), NOW()),

(7, 4, 1, NOW(), NOW()),

(7, 9, 2, NOW(), NOW()),

(8, 6, 1, NOW(), NOW()),

(9, 5, 1, NOW(), NOW()),

(10, 1, 1, NOW(), NOW()),

(10, 10, 2, NOW(), NOW());

-- historico\_reproducao

INSERT INTO historico\_reproducao

       ("id\_usuario", "id\_musica", "data\_hora", "createdAt", "updatedAt")

VALUES

(1, 2, '2025-06-10 08:30:00', NOW(), NOW()),

(1, 7, '2025-06-10 08:35:00', NOW(), NOW()),

(2, 5, '2025-06-09 22:00:00', NOW(), NOW()),

(3, 1, '2025-06-08 19:20:00', NOW(), NOW()),

(4, 3, '2025-06-07 15:45:00', NOW(), NOW()),

(5, 6, '2025-06-06 18:00:00', NOW(), NOW()),

(6, 7, '2025-06-05 21:15:00', NOW(), NOW()),

(7, 4, '2025-06-05 16:50:00', NOW(), NOW()),

(8, 6, '2025-06-04 14:30:00', NOW(), NOW()),

(9, 5, '2025-06-03 20:00:00', NOW(), NOW());

-- curtidas

INSERT INTO curtidas

       ("id\_usuario", "id\_musica", "data\_curtida", "createdAt", "updatedAt")

VALUES

(1, 2, '2025-06-10 09:00:00', NOW(), NOW()),

(1, 7, '2025-06-10 09:05:00', NOW(), NOW()),

(2, 5, '2025-06-09 22:10:00', NOW(), NOW()),

(3, 1, '2025-06-08 19:30:00', NOW(), NOW()),

(4, 3, '2025-06-07 15:50:00', NOW(), NOW()),

(5, 6, '2025-06-06 18:05:00', NOW(), NOW()),

(6, 7, '2025-06-05 21:20:00', NOW(), NOW()),

(7, 4, '2025-06-05 16:55:00', NOW(), NOW()),

(8, 6, '2025-06-04 14:35:00', NOW(), NOW()),

(9, 5, '2025-06-03 20:05:00', NOW(), NOW());

**Principais consultas mapeadas baseadas em regras de negócio (mínimo 4)**

ALTER TABLE musica

ADD COLUMN IF NOT EXISTS reproducoes\_total BIGINT DEFAULT 0;

CREATE OR REPLACE FUNCTION fn\_add\_musica\_playlist(

    p\_playlist\_id INT,

    p\_musica\_id   INT

) RETURNS VOID AS $$

DECLARE

    v\_next\_ordem INT;

BEGIN

    SELECT COALESCE(MAX(ordem),0)+1

      INTO v\_next\_ordem

      FROM playlist\_musica

     WHERE id\_playlist = p\_playlist\_id;

    INSERT INTO playlist\_musica (id\_playlist, id\_musica, ordem)

    VALUES (p\_playlist\_id, p\_musica\_id, v\_next\_ordem);

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION fn\_toggle\_curtida(

    p\_usuario\_id INT,

    p\_musica\_id  INT

) RETURNS TEXT AS $$

BEGIN

    IF EXISTS (SELECT 1

                 FROM curtidas

                WHERE id\_usuario = p\_usuario\_id

                  AND id\_musica  = p\_musica\_id) THEN

        DELETE FROM curtidas

         WHERE id\_usuario = p\_usuario\_id

           AND id\_musica  = p\_musica\_id;

        RETURN 'Curtida removida';

    ELSE

        INSERT INTO curtidas (id\_usuario, id\_musica, data\_curtida, "createdAt", "updatedAt")

        VALUES (p\_usuario\_id, p\_musica\_id, NOW(), NOW(), NOW());

        RETURN 'Curtida adicionada';

    END IF;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION fn\_top\_musicas\_usuario(

    p\_usuario\_id INT,

    p\_limite     INT DEFAULT 10

) RETURNS TABLE(

    id\_musica INT,

    titulo    VARCHAR,

    total\_reproducoes BIGINT

) AS $$

BEGIN

    RETURN QUERY

    SELECT m.id\_musica,

           m.titulo,

           COUNT(\*) AS total\_reproducoes

      FROM historico\_reproducao h

      JOIN musica m ON m.id\_musica = h.id\_musica

     WHERE h.id\_usuario = p\_usuario\_id

     GROUP BY m.id\_musica, m.titulo

     ORDER BY total\_reproducoes DESC

     LIMIT p\_limite;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION trgfn\_incrementa\_reproducoes()

RETURNS TRIGGER AS $$

BEGIN

    UPDATE musica

       SET reproducoes\_total = reproducoes\_total + 1

     WHERE id\_musica = NEW.id\_musica;

    RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_incrementa\_reproducoes

AFTER INSERT ON historico\_reproducao

FOR EACH ROW

EXECUTE PROCEDURE trgfn\_incrementa\_reproducoes();

CREATE OR REPLACE FUNCTION trgfn\_decrementa\_reproducoes()

RETURNS TRIGGER AS $$

BEGIN

  UPDATE musica

     SET reproducoes\_total = GREATEST(reproducoes\_total - 1, 0)

   WHERE id\_musica = OLD.id\_musica;

  RETURN OLD;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_decrementa\_reproducoes

AFTER DELETE ON historico\_reproducao

FOR EACH ROW

EXECUTE PROCEDURE trgfn\_decrementa\_reproducoes();

CREATE OR REPLACE FUNCTION trgfn\_ajusta\_reproducoes\_update()

RETURNS TRIGGER AS $$

BEGIN

  IF NEW.id\_musica <> OLD.id\_musica THEN

     UPDATE musica

        SET reproducoes\_total = GREATEST(reproducoes\_total - 1, 0)

      WHERE id\_musica = OLD.id\_musica;

     UPDATE musica

        SET reproducoes\_total = reproducoes\_total + 1

      WHERE id\_musica = NEW.id\_musica;

  END IF;

  RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_ajusta\_reproducoes

AFTER UPDATE OF id\_musica ON historico\_reproducao

FOR EACH ROW

EXECUTE PROCEDURE trgfn\_ajusta\_reproducoes\_update();

CREATE OR REPLACE FUNCTION fn\_clone\_playlist(

  p\_playlist\_id  INT,

  p\_novo\_user\_id INT,

  p\_novo\_nome    TEXT

) RETURNS INT AS $$

DECLARE

  v\_new\_id INT;

BEGIN

  INSERT INTO playlist (nome, id\_usuario, "createdAt", "updatedAt")

       VALUES (p\_novo\_nome, p\_novo\_user\_id, NOW(), NOW())

  RETURNING id\_playlist INTO v\_new\_id;

  INSERT INTO playlist\_musica (id\_playlist, id\_musica, ordem)

  SELECT v\_new\_id, id\_musica, ordem

    FROM playlist\_musica

   WHERE id\_playlist = p\_playlist\_id

ORDER BY ordem;

  RETURN v\_new\_id;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION fn\_move\_musica\_playlist(

  p\_playlist\_id INT,

  p\_musica\_id   INT,

  p\_nova\_ordem  INT

) RETURNS VOID AS $$

DECLARE

  v\_ordem\_atual INT;

BEGIN

  SELECT ordem

    INTO v\_ordem\_atual

    FROM playlist\_musica

   WHERE id\_playlist = p\_playlist\_id

     AND id\_musica   = p\_musica\_id;

  IF NOT FOUND OR v\_ordem\_atual = p\_nova\_ordem THEN

     RETURN;

  END IF;

  IF v\_ordem\_atual < p\_nova\_ordem THEN

     UPDATE playlist\_musica

        SET ordem = ordem - 1

      WHERE id\_playlist = p\_playlist\_id

        AND ordem BETWEEN v\_ordem\_atual + 1 AND p\_nova\_ordem;

  ELSE

     UPDATE playlist\_musica

        SET ordem = ordem + 1

      WHERE id\_playlist = p\_playlist\_id

        AND ordem BETWEEN p\_nova\_ordem AND v\_ordem\_atual - 1;

  END IF;

  UPDATE playlist\_musica

     SET ordem = p\_nova\_ordem

   WHERE id\_playlist = p\_playlist\_id

     AND id\_musica   = p\_musica\_id;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION fn\_clear\_playlist(p\_playlist\_id INT)

RETURNS INT AS $$

DECLARE

  v\_rows INT;

BEGIN

  DELETE FROM playlist\_musica

   WHERE id\_playlist = p\_playlist\_id;

  GET DIAGNOSTICS v\_rows = ROW\_COUNT;

  RETURN v\_rows;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION fn\_user\_resumo(p\_usuario\_id INT)

RETURNS TABLE (

  total\_playlists   INT,

  total\_curtidas    INT,

  total\_reproducoes BIGINT

) AS $$

BEGIN

  RETURN QUERY

  SELECT

    (SELECT COUNT(\*) FROM playlist             WHERE id\_usuario = p\_usuario\_id),

    (SELECT COUNT(\*) FROM curtidas             WHERE id\_usuario = p\_usuario\_id),

    (SELECT COUNT(\*) FROM historico\_reproducao WHERE id\_usuario = p\_usuario\_id);

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE VIEW vw\_playlist\_detalhes AS

SELECT p.id\_playlist,

       p.nome               AS nome\_playlist,

       u.nome               AS nome\_usuario,

       COUNT(pm.id\_musica)  AS qtd\_musicas

  FROM playlist p

  JOIN usuario u        ON u.id\_usuario = p.id\_usuario

  LEFT JOIN playlist\_musica pm

       ON pm.id\_playlist = p.id\_playlist

 GROUP BY p.id\_playlist, p.nome, u.nome;

CREATE OR REPLACE VIEW vw\_musica\_completa AS

SELECT m.id\_musica,

       m.titulo                   AS titulo\_musica,

       m.genero,

       al.titulo                  AS titulo\_album,

       ar.nome                    AS nome\_artista,

       m.reproducoes\_total,

       (SELECT COUNT(\*)

          FROM curtidas c

         WHERE c.id\_musica = m.id\_musica) AS total\_curtidas

  FROM musica  m

  LEFT JOIN album   al ON al.id\_album   = m.id\_album

  LEFT JOIN artista ar ON ar.id\_artista = m.id\_artista;

CREATE OR REPLACE VIEW vw\_top10\_mais\_tocadas AS

SELECT m.id\_musica,

       m.titulo,

       ar.nome              AS artista,

       m.reproducoes\_total

  FROM musica  m

  JOIN artista ar ON ar.id\_artista = m.id\_artista

 ORDER BY m.reproducoes\_total DESC

 LIMIT 10;

CREATE INDEX IF NOT EXISTS idx\_album\_id\_artista                ON album(id\_artista);

CREATE INDEX IF NOT EXISTS idx\_musica\_id\_artista               ON musica(id\_artista);

CREATE INDEX IF NOT EXISTS idx\_playlist\_id\_usuario             ON playlist(id\_usuario);

CREATE INDEX IF NOT EXISTS idx\_playlist\_musica\_id\_musica       ON playlist\_musica(id\_musica);

CREATE INDEX IF NOT EXISTS idx\_historico\_reproducao\_id\_musica  ON historico\_reproducao(id\_musica);

CREATE INDEX IF NOT EXISTS idx\_historico\_reproducao\_id\_usuario ON historico\_reproducao(id\_usuario);

CREATE INDEX IF NOT EXISTS idx\_curtidas\_id\_musica              ON curtidas(id\_musica);

CREATE UNIQUE INDEX IF NOT EXISTS idx\_usuario\_email            ON usuario(email);

SELECT fn\_clone\_playlist(1, 2, 'Clone da lista 1') AS nova\_playlist;

SELECT fn\_move\_musica\_playlist( (SELECT MAX(id\_playlist) FROM playlist) , 1, 3 );

SELECT fn\_clear\_playlist( (SELECT MAX(id\_playlist) FROM playlist) ) AS removidas;

INSERT INTO historico\_reproducao

       (id\_usuario, id\_musica, data\_hora, "createdAt", "updatedAt")

VALUES (1, 1, NOW(), NOW(), NOW());

DELETE FROM historico\_reproducao

 WHERE id\_usuario = 1

   AND id\_musica  = 1

 LIMIT 1;

SELECT id\_musica, reproducoes\_total FROM musica WHERE id\_musica = 1;