

PROJETO DATABASE SPOTIFY CLONE

[desafio1.sql](#)

Desafio 1

Criação do Banco de Dados

Inicialmente, criaremos um banco de dados denominado "**SpotifyClone**". Este banco servirá como o núcleo onde guardaremos todos os dados referentes a artistas, álbuns, músicas, usuários, playlists e as conexões entre essas entidades.

Criando o banco de dados com o nome "**SpotifyClone**" através da consulta:

```
CREATE DATABASE SpotifyClone;
```

Criação das Tabelas Normalizadas

O próximo passo consiste na elaboração de tabelas normalizadas. Isso implica que vamos organizar os dados em entidades separadas, de forma eficaz e sem redundância. As tabelas seguirão a estrutura normalizada para assegurar a

correta armazenagem das informações, prevenindo inconsistências e simplificando consultas futuras.

Tabelas:

- **Usuários:** Contém informações dos usuários.
- **Artistas:** Contém os artistas.
- **Álbuns:** Contém os álbuns e as associações com artistas.
- **Músicas:** Contém as músicas e suas associações com álbuns e artistas.
- **Histórico de Reprodução:** Contém o histórico de playlists reproduzidas pelos usuários.
- **Seguindo:** Contém artistas e usuários que estão seguindo.

Cada tabela será estabelecida com suas chaves primárias e estrangeiras correspondentes para assegurar a consistência referencial entre as entidades.

```
USE SpotifyClone;

-- Criando a tabela de usuários
CREATE TABLE usuarios (
    id_usuario INT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(90),
    email VARCHAR(90) UNIQUE,
    data_nascimento DATE,
    plano VARCHAR(50)
);

-- Criando a tabela de artistas
CREATE TABLE artistas (
    id_artista INT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(90)
);

-- Criando a tabela de álbuns
CREATE TABLE albuns (
    id_album INT PRIMARY KEY AUTO_INCREMENT,
    titulo VARCHAR(90),
    id_artista INT,
```

```

        ano_lancamento YEAR,
        FOREIGN KEY (id_artista) REFERENCES artistas(id_artista)
    );

-- Criando a tabela de músicas
CREATE TABLE musicas (
    id_musica INT PRIMARY KEY AUTO_INCREMENT,
    titulo VARCHAR(90),
    duracao TIME,
    id_album INT,
    FOREIGN KEY (id_album) REFERENCES albuns(id_album)
);

-- Criando a tabela de histórico de reprodução
CREATE TABLE historico_reproducao (
    id_reproducao INT PRIMARY KEY AUTO_INCREMENT,
    id_usuario INT,
    id_musica INT,
    data_reproducao DATETIME,
    FOREIGN KEY (id_usuario) REFERENCES usuarios(id_usuario),
    FOREIGN KEY (id_musica) REFERENCES musicas(id_musica)
);

-- Criando a tabela de pessoas seguindo artistas
CREATE TABLE seguindo_artistas (
    id_usuario INT,
    id_artista INT,
    PRIMARY KEY (id_usuario, id_artista),
    FOREIGN KEY (id_usuario) REFERENCES usuarios(id_usuario),
    FOREIGN KEY (id_artista) REFERENCES artistas(id_artista)
);

```

Inserção de Dados nas Tabelas

Depois de criar as tabelas, a próxima etapa é "encher" essas tabelas com dados autênticos. Isso implica a adição de dados referentes a artistas, álbuns, canções, usuários e registros históricos.

A inserção dos dados será feita conforme a estrutura das tabelas, preservando a integridade das relações.

```
-- Inserindo dados na tabela de usuários
INSERT INTO usuarios (nome, email, data_nascimento, plano) VALUES
('Maria Corrales', 'maria@gmail.com', '2005-04-08', 'Premium'),
('Guido Fernandes', 'guido@gmail.com', '1988-11-23', 'Free');

-- Inserindo dados na tabela de artistas
INSERT INTO artistas (nome) VALUES
('Artista 1'),
('Artista 2');

-- Inserindo dados na tabela de álbuns
INSERT INTO albums (titulo, id_artista, ano_lancamento) VALUES
('Album 1', 1, 2020),
('Album 2', 2, 2021);

-- Inserindo dados na tabela de músicas
INSERT INTO musicas (titulo, duracao, id_album) VALUES
('Musica 1', '00:03:45', 1),
('Musica 2', '00:04:20', 2);

-- Inserindo dados na tabela de histórico de reprodução
INSERT INTO historico_reproducao (id_usuario, id_musica, data) VALUES
(1, 1, '2023-09-01 10:00:00'),
(2, 2, '2023-09-02 15:30:00');

-- Inserindo dados na tabela de pessoas seguindo artistas
INSERT INTO seguindo_artistas (id_usuario, id_artista) VALUES
(1, 1),
(2, 2);
```

Criação do Arquivo de Configurações "desafio1.json"

Vamos criar um arquivo de configuração denominado "**desafio1.json**" para permitir uma avaliação automática do nosso desafio. Este documento identifica

as tabelas e colunas que contêm os dados cruciais para o procedimento de avaliação.

Formato:

O arquivo seguirá um formato específico, que inclui a definição de cada tabela e suas respectivas colunas.

```
{
  "usuario": {
    "coluna_usuario": "nome",
    "tabela_que_contem_usuario": "usuarios"
  },
  "artista": {
    "coluna_artista": "nome",
    "tabela_que_contem_artista": "artistas"
  },
  "album": {
    "coluna_album": "titulo",
    "tabela_que_contem_album": "albuns"
  },
  "musica": {
    "coluna_musica": "titulo",
    "tabela_que_contem_musica": "musicas"
  },
  "historico_reproducao": {
    "coluna_historico": "data_reproducao",
    "tabela_que_contem_historico": "historico_reproducao"
  },
  "seguindo_artistas": {
    "coluna_usuario_seguindo": "id_usuario",
    "coluna_artista_seguindo": "id_artista",
    "tabela_que_contem_seguindo": "seguindo_artistas"
  }
}
```

desafio1.json

Desafio 2

Criação da "View" "estatisticas_musicais"

A próxima etapa consiste em criar uma "View" denominada "estatisticas_musicais". Uma "View" é um tipo de tabela virtual que facilita consultas complexas ao combinar informações de uma ou mais tabelas. Esta análise será fundamental para produzir estatísticas relevantes sobre o nosso banco de dados musical.

Objetivos da "View" "estatisticas_musicais"

A "View" será composta por três colunas, cada uma representando uma métrica crucial:

1. Quantidade Total de Canções:

- Esta coluna mostrará a quantidade total de músicas registradas no banco de dados.
- Vamos chamar esta coluna de **cancoes**.

2. Quantidade Total de Artistas:

- Esta coluna mostrará a quantidade total de artistas que foram registrados.
- Será atribuído o alias **artistas**.

3. Quantidade Total de Álbuns:

- Esta coluna exibirá a quantidade total de álbuns armazenados no banco de dados.
- O alias será "albuns".

Criando a "View"

A criação da "View" será feita por meio de uma consulta na base de dados SQL que realiza contagens a partir das tabelas correspondentes.

```
-- Criando a VIEW estatisticas_musicais
CREATE VIEW estatisticas_musicais AS
SELECT
    (SELECT COUNT(*) FROM musicas) AS cancoes,
```

```
(SELECT COUNT(*) FROM artistas) AS artistas,  
(SELECT COUNT(*) FROM albuns) AS albuns;
```

Verificando a "View"

Depois de criar a "View", é crucial verificar se ela foi corretamente criada e se apresenta os dados esperados. Isso inclui:

- Confirmar a existência da "View" "**estatisticas_musicais**" no banco de dados.
- Consultar a "View" para confirmar que as colunas **cancoes**, **artistas** e **albuns** exibem os valores corretos, conforme os dados que inserimos anteriormente.

```
-- Consultando a VIEW estatisticas_musicais  
SELECT * FROM estatisticas_musicais;
```

	cancoes	artistas	albuns
▶	2	2	2

Desafio 3

Criação da "View" "historico_reproducao_usuarios"

Em nosso projeto, o próximo passo é a criação de uma nova "View" denominada **historico_reproducao_usuarios**. Esta "View" nos possibilitará monitorar o histórico de reprodução dos usuários, facilitando a visualização das músicas que cada um escutou.

Objetivos da "View" "historico_reproducao_usuarios"

A "View" terá duas colunas principais que oferecerão dados relevantes:

1. Usuário:

- Esta coluna apresentará o nome do usuário.
- Daremos a essa coluna o nome de **usuario**.

2. Nome:

- Esta coluna mostrará o nome da música que a pessoa escutou com base no seu histórico de reprodução.
- O alias atribuído será **nome**.

Implementando da "View"

Uma "View" é criada combinando tabelas contendo informações sobre usuários e as músicas que eles tocaram.

```
-- Criando a VIEW historico_reproducao_usuarios
CREATE VIEW historico_reproducao_usuarios AS
SELECT
    u.nome AS usuario,
    m.titulo AS nome
FROM
    historico_reproducao hr
JOIN
    usuarios u ON hr.id_usuario = u.id_usuario
JOIN
    musicas m ON hr.id_musica = m.id_musica
ORDER BY
    u.nome ASC,
    m.titulo ASC;
```

Verificando a VIEW

Após criar uma "View", verificamos se ela foi criada corretamente e retorna os dados esperados. Isso incluirá:

- Confirmar a existência da "View" "**historico_reproducao_usuarios**" no banco de dados.
- Consultar a "View" para confirmar que os valores das colunas **usuario** e **nome** estão corretos.
- Confirmar se os resultados estão devidamente organizados:
 - Primeiro por nome da pessoa usuária em ordem alfabética.
 - Em caso de empate, por nome da canção em ordem alfabética.


```
-- Consultando a VIEW historico_reproducao_usuarios
SELECT * FROM historico_reproducao_usuarios;
```

	usuario	nome
▶	Guido Fernandes	Musica 2
	Maria Corrales	Musica 1

Desafio 4

Criação da "View" "top_3_artistas"

Em nosso projeto, o próximo passo é a criação de uma nova "View" denominada **top_3_artistas**. Essa View" será importante para realçar os artistas mais populares no nosso banco de dados "**SpotifyClone**", possibilitando aos usuários uma rápida visualização dos artistas mais populares.

Objetivos da "View" "top_3_artistas"

A "View" terá duas colunas principais:

1. Artista:

- Esta coluna apresentará o nome da pessoa artista.
- Vamos atribuir a esta coluna o apelido de **artista**.

2. Seguidores:

- Esta coluna mostrará a quantidade de pessoas que estão seguindo aquele artista.
- O alias atribuído será **seguidores**.

Gerando a "View"

A "View" será gerada através de um consulta no banco de dados SQL que identifica os três artistas mais influentes com base na quantidade de seguidores.

```
-- Criando a VIEW top_3_artistas
CREATE VIEW top_3_artistas AS
SELECT
    a.nome AS artista,
    COUNT(sa.id_usuario) AS seguidores
FROM
    artistas a
JOIN
    seguindo_artistas sa ON a.id_artista = sa.id_artista
GROUP BY
    a.nome
ORDER BY
    seguidores DESC,
    a.nome ASC
LIMIT 3;
```

Verificando a "View"

Depois de criarmos a "View", é essencial verificarmos se ela foi corretamente criada e se apresenta os dados esperados. Isso inclui:

- Confirmar a existência da "View" "**top_3_artistas**" no banco de dados.
- Consultar a "View" para confirmar que os valores das colunas **artista** e **seguidores** estão corretos.
- Confirmar se os resultados estão devidamente organizados:
 - Primeiro pela quantidade de seguidores em ordem decrescente.
 - Em caso de empate, pelo nome do artista em ordem alfabética.

```
-- Consultando a VIEW top_3_artistas
SELECT * FROM top_3_artistas;
```

	artista	seguidores
▶	Artista 1	1
	Artista 2	1

Desafio 5

Criação da "View" "top_2_hits_do_momento"

Como parte do nosso projeto, vamos nos concentrar na análise das músicas mais populares da atualidade. Portanto, vamos criar uma "View" denominada **top_2_hits_do_momento**, que nos possibilitará reconhecer de forma rápida as duas músicas mais tocadas na plataforma.

Objetivos da "View" "top_2_hits_do_momento"

A VIEW terá duas colunas essenciais:

1. Canção:

- Esta coluna mostrará o nome da canção.
- O alias **cancao** será atribuído.

2. Reproduções:

- Esta coluna mostrará a quantidade de pessoas que já escutaram a canção.
- O alias para essa coluna será **reproducoes**.

Implementação da VIEW

A "View" será gerada por meio de uma consulta SQL que determinara as duas músicas mais executadas, considerando o número de execuções.

```
-- Criando a VIEW top_2_hits_do_momento
CREATE VIEW top_2_hits_do_momento AS
SELECT
    m.titulo AS cancao,
    COUNT(hr.id_usuario) AS reproducoes
FROM
    musicas m
JOIN
    historico_reproducao hr ON m.id_musica = hr.id_musica
GROUP BY
    m.titulo
ORDER BY
    reproducoes DESC,
```

```
m.titulo ASC
LIMIT 2;
```

Verificando a "View"

Depois de criar a "View", é fundamental realizar algumas verificações para garantir que ela funcione conforme o esperado. As validações incluirão:

- Confirmar se a "View" **"top_2_hits_do_momento"** foi devidamente inserida no sistema.
- Confirmar na "View" se as colunas **cancao** e **reproducoes** estão apresentando os valores adequados.
- Confirmar se os resultados estão organizados de maneira adequada:
 - Primeiro, pela quantidade de reproduções em ordem decrescente.
 - Em caso de empate, pelo nome da canção em ordem alfabética.

```
-- Consultando a VIEW top_2_hits_do_momento
SELECT * FROM top_2_hits_do_momento;
```

	cancao	reproducoes
▶	Musica 1	1
	Musica 2	1

Desafio 6

Criando a Tabela "planos" e inserindo dados

```
-- Exemplo de criação da tabela planos
CREATE TABLE planos (
    nome_plano VARCHAR(50) PRIMARY KEY,
    valor DECIMAL(10, 2)
);

-- Exemplo de inserção de dados na tabela planos
INSERT INTO planos (nome_plano, valor) VALUES
```

```
('Free', 0.00),  
('Premium', 29.90);
```

Criação da "View" "faturamento_atual"

Prosseguindo com o nosso projeto, a próxima etapa será a criação de uma "View" que trará dados cruciais sobre a receita da empresa, considerando os planos de assinatura dos nossos usuários. Esta "View" será identificada como **faturamento_atual** e visa consolidar informações financeiras cruciais para a avaliação do desempenho da plataforma.

Objetivos da "View" "faturamento_atual"

A "View" "faturamento_atual" apresentará quatro colunas essenciais:

1. Faturamento Mínimo:

- Esta coluna mostrará o menor valor de plano disponível para um usuário.
- O alias atribuído será **faturamento_minimo**.

2. Faturamento Máximo:

- Esta coluna mostrará o maior valor de plano existente para uma pessoa usuária.
- O alias será **faturamento_maximo**.

3. Faturamento Médio:

- Aqui, apresentaremos o valor médio dos planos que os usuários possuem até o momento.
- O alias dessa coluna será **faturamento_medio**.

4. Faturamento Total:

- Esta coluna mostrará o valor total obtido pelos planos detidos pelos usuários.
- O alias atribuído será **faturamento_total**.

Implementação da "View"

Utilizaremos um comando SQL para criar a "View", que realizará as seguintes operações:

- Calcular o menor, o maior, o médio e o total dos valores dos planos, garantindo que os resultados sejam ajustados para duas casas decimais;

```
-- Criando a VIEW faturamento_atual
CREATE VIEW faturamento_atual AS
SELECT
    ROUND(MIN(p.valor), 2) AS faturamento_minimo,
    ROUND(MAX(p.valor), 2) AS faturamento_maximo,
    ROUND(AVG(p.valor), 2) AS faturamento_medio,
    ROUND(SUM(p.valor), 2) AS faturamento_total
FROM
    usuarios u
JOIN
    planos p ON u.plano = p.nome_plano;
```

Verificando a “

Depois de criar a “View” “**faturamento_atual**”, serão feitas as seguintes validações:

- Confirmar se a “View” “**faturamento_atual**” foi criada com sucesso no banco de dados.
- Consultar a “View” para confirmar que as colunas **faturamento_minimo**, **faturamento_maximo**, **faturamento_medio** e **faturamento_total** estão corretos.
- Garantir que os valores foram corretamente arredondados para duas casas decimais.

```
-- Consultando a VIEW faturamento_atual
SELECT * FROM faturamento_atual;
```

	faturamento_minimo	faturamento_maximo	faturamento_medio	faturamento_total
▶	0.00	29.90	14.95	29.90

Desafio 7

Criação da “View” “perfil_artistas”

Na próxima parte do projeto vamos nos concentrar na criação de uma "View" que mostre todos os livros que cada artista fez e quantos seguidores eles têm. Esta "View" chamada "perfil_artistas" é necessária para conhecer a popularidade e atividades dos artistas em nossa plataforma..

A "View" "**perfil_artistas**" mostra três colunas principais:

1. Artista:

- Esta coluna mostra o nome do artista.
- O nome de usuário especificado é **art**.

1. Álbum:

- O nome do álbum feito pelo artista é exibido aqui.
- O nome desta coluna é **Álbum**.

1. Seguidores:

- Esta coluna mostra quantas pessoas seguem o artista.
 - O apelido é **Seguidor..**

Objetivos da "View" "perfil_artistas"

A "View" "**perfil_artistas**" mostra três colunas principais:

1. Artista:

- Esta coluna mostra o nome do artista.
- O nome de usuário especificado é **art**.

2. Álbum:

- O nome do álbum feito pelo artista é exibido aqui.
- O nome desta coluna é **album**.

3. Seguidores:

- Esta coluna mostra quantas pessoas seguem o artista.
- O alias será **seguidores**

Implementação da VIEW

Para criar a "View", utilizaremos uma solicitação SQL que reunirá os dados de artistas, álbuns e seguidores. A consulta irá ordenar os resultados conforme os

critérios estabelecidos:

- Ordenação em ordem decrescente pelo número de seguidores.
- Em caso de empate, por nome do artista em ordem alfabética.
- Para artistas com o mesmo nome, por nome do álbum em ordem alfabética.

```
-- Removendo a VIEW existente
DROP VIEW IF EXISTS perfil_artistas;

-- Criando a VIEW novamente
CREATE VIEW perfil_artistas AS
SELECT
    a.nome AS artista,
    al.titulo AS album,
    COUNT(sa.id_usuario) AS seguidores
FROM
    artistas a
JOIN
    albums al ON a.id_artista = al.id_artista
LEFT JOIN
    seguindo_artistas sa ON a.id_artista = sa.id_artista
GROUP BY
    a.nome, al.titulo
ORDER BY
    seguidores DESC,
    a.nome ASC,
    al.titulo ASC;
```

Validação da "View"

Após a criação da "View" "**perfil_artistas**", realizamos as seguintes validações:

- Confirmamos se a "View" "**perfil_artistas**" foi criada corretamente no banco de dados.
- Consultamos a "View" para garantir que as colunas **artista**, **album** e **seguidores** estão retornando os dados corretos.
- Verificamos a ordem dos resultados conforme os critérios estabelecidos.


```
-- Consultando a VIEW existente
SELECT * FROM perfil_artistas;
```

	artista	album	seguidores
▶	Artista 1	Album 1	1
	Artista 2	Album 2	1

Desafio 8

Criação da Trigger "trigger_usuario_delete"

À medida que avançamos em nosso projeto, a próxima etapa é implementar uma **"Trigger"** que garanta a integridade dos dados quando um usuário for excluído. O nome desta "Trigger" é **"trigger_usuario_delete"** e sua função é garantir que a exclusão do usuário seja detectada automaticamente em todas as tabelas relacionadas.

Objetivos da Trigger "trigger_usuario_delete"

A "Trigger" **"trigger_usuario_delete"** será configurada para disparar sempre que um usuário for excluído do banco de dados. Seu objetivo é:

- Remover todas as referências ao usuário nas tabelas associadas, garantindo que não haja dados órfãos.

Implementação da Trigger

A "Trigger" será implementada com uma consulta SQL que realizará a seguinte operação:

- Ao detectar a exclusão de uma pessoa usuária, todas as entradas relacionadas em outras tabelas serão removidas.

Mudando o delimitador para evitar conflitos:

```
-- Mudando o delimitador para evitar conflitos com o uso de '
DELIMITER $$

-- Criando a trigger para deletar registros relacionados ao u
CREATE TRIGGER trigger_usuario_delete
```

```

BEFORE DELETE ON usuarios
FOR EACH ROW
BEGIN
    -- Excluindo o histórico de reprodução do usuário
    DELETE FROM historico_reproducao WHERE id_usuario = OLD.id_usuario;

    -- Excluindo os registros de artistas seguidos pelo usuário
    DELETE FROM seguindo_artistas WHERE id_usuario = OLD.id_usuario;

    -- Adicionando mais comandos de exclusão conforme necessário
    -- DELETE FROM outra_tabela WHERE id_usuario = OLD.id_usuario;

END;
$$

-- Retornando o delimitador ao padrão
DELIMITER ;

```

Teste da Funcionalidade

Após a criação da "Trigger", realizamos um teste para garantir seu funcionamento. Para isso, seguiremos os seguintes passos:

- 1 - Desabilitar o modo seguro temporariamente (SQL_SAFE_UPDATES = 0)
- 2 - Executar a exclusão da usuária chamada **"Maria Corrales"**.
- 3 - Verificar se todas as entradas relacionadas a essa usuária foram removidas das tabelas associadas.
- 4 - Habilitar o modo seguro novamente (SQL_SAFE_UPDATES = 1)

```

-- Desativando o safe update mode para esta sessão
SET SQL_SAFE_UPDATES = 0;
-- Excluindo a usuária "Thati"
DELETE FROM usuarios WHERE nome = 'Maria Corrales';
-- Ativando novamente o safe update mode
SET SQL_SAFE_UPDATES = 1;

```

Em seguida, realizamos consultas nas tabelas que deveriam ser afetadas:

```
-- Verificando se o usuário "Maria Corrales" foi excluído de
SELECT * FROM historico_reproducao WHERE id_usuario = (SELECT
SELECT * FROM seguindo_artistas WHERE id_usuario = (SELECT id
```

	id_usuario	id_artista
	NULL	NULL

Desafio 9

Criação da "Procedure" "albuns_do_artista"

Avançando em nosso projeto, o próximo passo é a implementação de uma "Procedure" **albuns_do_artista**. Esta "Procedure" será responsável por retornar informações sobre os álbuns de um artista específico, facilitando a consulta de dados no banco.

Objetivos da "Procedure" "albuns_do_artista"

A "Procedure" **albuns_do_artista** receberá como parâmetro o nome de um artista e deverá retornar as seguintes colunas:

1. O nome do artista, com o alias **"artista"**.
2. O nome do álbum, com o alias **"album"**.

Implementação da "Procedure"

A "Procedure" será implementada com um comando SQL que realizará a seleção dos álbuns associados ao artista fornecido como parâmetro. O resultado será ordenado pelo nome do álbum em ordem alfabética.

```
-- Mudando o delimitador para criar a procedure
DELIMITER $$

-- Criando a procedure albuns_do_artista
CREATE PROCEDURE albuns_do_artista(IN nome_artista VARCHAR(100))
BEGIN
    -- Selecionando o nome do artista e seus álbuns, ordenado
```

```

SELECT
    a.nome AS artista,
    al.titulo AS album
FROM
    artistas a
JOIN
    albums al ON a.id_artista = al.id_artista
WHERE
    a.nome = nome_artista
ORDER BY
    al.titulo ASC;
END;
$$

-- Retornando o delimitador ao padrão
DELIMITER ;

```

Teste da "Procedure"

Após a criação da "Procedure", realizamos um teste para garantir seu funcionamento correto. Chamamos a "Procedure" passando o nome **"artista"** como parâmetro.

```

-- Chamando a procedure para testar com "Album 1"
CALL albums_do_artista('Artista 1');

```

	artista	album
▶	Artista 1	Album 1

Desafio 10

Criação da "Function" "quantidade_musicas_no_historico"

Dando continuidade ao desenvolvimento do nosso banco de dados, agora iremos implementar uma **"function"** chamada

"quantidade_musicas_no_historico" Esta função será fundamental para obter

rapidamente a quantidade de músicas que um usuário possui em seu histórico de reprodução.

Objetivos da "Function" "quantidade_musicas_no_historico"

A function "quantidade_musicas_no_historico" será responsável por:

- Receber como parâmetro o código identificador da pessoa usuária.
- Retornar a quantidade de canções presentes em seu histórico de reprodução.

Implementação da "Function"

A implementação da "function" será feita utilizando uma consulta no banco de dados SQL que contará as músicas no histórico da pessoa usuária identificada pelo seu ID.

```
-- Mudando o delimitador para criar a function
DELIMITER $$

-- Criando a function quantidade_musicas_no_historico
CREATE FUNCTION quantidade_musicas_no_historico(id_usuario INT)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE qtd_musicas INT;

    -- Selecionando a quantidade de músicas no histórico de reprodução
    SELECT COUNT(*) INTO qtd_musicas
    FROM historico_reproducao
    WHERE id_usuario = id_usuario;

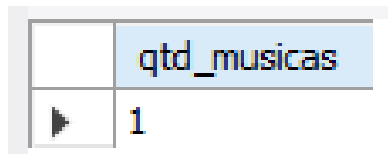
    -- Retornando a quantidade de músicas
    RETURN qtd_musicas;
END;
$$

-- Retornando o delimitador ao padrão
DELIMITER ;
```

Testando a "Function"

Após a criação da "function", realizamos um teste para assegurar seu funcionamento. Chamamos a "function" passando o ID da pessoa usuária cujo nome é **"Guido Fernandes"**.

```
-- Chamando a função com o id de Bill
SELECT quantidade_musicas_no_historico ((SELECT id_usuario FR
```



	qtd_musicas
▶	1

Desafio 11

Para a criação da "View" "cancoes_premium" será necessário a criação da tabela "cancoes":

```
-- Criando a tabela cancoes
CREATE TABLE cancoes (
    id_cancao INT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(100) NOT NULL,
    duracao TIME, -- Duração da canção (opcional)
    id_album INT, -- Relacionamento com a tabela albuns
    CONSTRAINT fk_album FOREIGN KEY (id_album) REFERENCES alb
);

-- Inserindo dados na tabela cancoes
INSERT INTO cancoes (nome, duracao, id_album) VALUES
('Canção 1', '00:03:45', 1),
('Canção 2', '00:04:12', 1),
('Canção 3', '00:02:58', 2);
```

Vamos inserir colunas ainda inexistentes nas tabelas: "planos", "usuarios" e "histórico_reproducao", para evitar possíveis conflitos.

```
-- Alterando a chave primaria da tabela planos para a criação
ALTER TABLE planos
```

```

DROP PRIMARY KEY;
-- Criando a coluna "id_plano" e definindo ela como chave pri
ALTER TABLE planos
ADD COLUMN id_plano INT AUTO_INCREMENT PRIMARY KEY;

-- Adicionando "id_plano" na tabela "usuarios" para referenci
ALTER TABLE usuarios
ADD COLUMN id_plano INT,
ADD CONSTRAINT fk_plano FOREIGN KEY (id_plano) REFERENCES pla

-- Adicionando "nome_plano" na tabela "usuarios"
ALTER TABLE usuarios
ADD COLUMN nome_plano varchar(45);

-- Adicionando "id_cancao" na tabela "historico_reproducao"
ALTER TABLE historico_reproducao
ADD COLUMN id_cancao INT;

```

Criando a "View" "cancoes_premium"

Agora, vamos criar uma "View" chamada "**cancoes_premium**", cujo objetivo é mostrar o nome das canções e a quantidade de vezes que cada canções foi executada por usuários com os planos "familiar" ou "universitário". Esta metodologia possibilita uma análise mais precisa das canções mais populares entre esses grupos específicos de usuários.

1. Estrutura do JOIN (junção):

- A tabela "**historico_reproducao**" (também conhecida como "**hr**") é unida à tabela "**usuarios**" (denotada como "**u**") através da coluna "**id_usuario**". Essa relação é fundamental para associar as reproduções aos usuários.
- Em seguida, a tabela "**usuarios**" é unida com a tabela "**planos**" (identificada como "**p**") por meio da coluna "**nome_plano**". Isso possibilita identificar qual plano cada usuário tem.
- Por fim, a tabela "**historico_reproducao**" é vinculada à tabela "**cancoes**" (também conhecida como "**c**") por meio da coluna

"id_cancao", estabelecendo a conexão entre as reproduções e as canções correspondentes.

2. Filtragem de Dados:

- A expressão "WHERE" p.nome_plano IS ('Familiar', 'Universitário')" é usada para garantir que somente os usuários dos planos "Familiar" ou "Universitário" sejam levados em conta na contagem de reproduções.

3. Agrupamento e Contagem:

- Utilizamos a função "COUNT(hr.id_usuario)" para registrar a quantidade de execuções de cada canção. A cláusula "GROUP BY" c.nome" agrupa os resultados pelo nome da canção, proporcionando uma visão unificada das execuções.

4. Ordenação dos Resultados:

- Incluímos a cláusula "ORDER BY c.nome ASC" para classificar os resultados de maneira alfabética pelo nome da música, simplificando a análise.

```
CREATE VIEW cancoes_premium AS
SELECT
    c.nome AS nome,                                -- Nome da can
    COUNT(hr.id_usuario) AS reproducoes            -- Quantidade
FROM
    historico_reproducao hr                        -- Tabela do h
JOIN
    usuarios u ON hr.id_usuario = u.id_usuario    -- Relaciona o
JOIN
    planos p ON u.nome_plano = p.nome_plano        -- Relaciona o
JOIN
    cancoes c ON hr.id_cancao = c.id_cancao        -- Relaciona o
WHERE
    p.nome_plano IN ('Familiar', 'Universitário') -- Filtra u
GROUP BY
    c.nome                                           -- Agrupa os
ORDER BY
    c.nome ASC;                                     -- Ordena em

SHOW TABLES;
```



```
DESCRIBE historico_reproducao;
```

Testando a "View"

Depois de criar a "View", é possível testá-la através da seguinte consulta:

```
-- Consultando a view cancoes_premium  
SELECT * FROM cancoes_premium;
```

	nome	reproducoes
--	------	-------------

Este projeto foi desenvolvido por **Maria Fernanda Ribeiro Corrales e Guido Fernandes da Guarda**. Trata-se do Projeto Final De Banco De Dados do curso de Administrador de Banco de Dados, ofertado pelo SENAI - Taguatinga - DF em parceria com o Programa DF INOVATECH. Ministrado pela Professora Mirka Juliet.