Pró-reitora Acadêmica
Escola de Educação, Tecnologia e Comunicação
Curso de Bacharelado em Ciência da Computação

Modelagem e Implementação de Banco de Dados para Rede Social

Autores: Gabriel de Sousa Figueredo, Ingrid de Oliveira Bonifácio, Caio Henrique Pereira, Ytalo Santana Ribeiro, Cleber Duarte Silva

Professor: João Robson Martins

Sumário

Relatório Técnico

1. Introdução

O objetivo deste trabalho foi criar um banco de dados para gerenciar informações de uma rede social, com foco na interação entre usuários, postagens, grupos e avaliações. O banco foi projetado para armazenar e organizar dados relacionados a usuários, conexões entre eles, grupos aos quais pertencem, postagens e comentários, além de avaliações feitas sobre postagens e comentários.

A estrutura do banco de dados permite um gerenciamento eficiente das interações sociais, possibilitando consultas detalhadas sobre conexões entre usuários, atividades realizadas (postagens, comentários), e engajamento com o conteúdo por meio de notificações e avaliações.

Os dados foram modelados com base em um cenário fictício, mas representam situações típicas encontradas em sistemas de redes sociais modernas. As tabelas foram desenhadas para refletir relações entre os usuários e suas atividades na plataforma, garantindo flexibilidade e escalabilidade para futuras expansões do sistema.

2. Estrutura do Banco de Dados

O banco de dados contém 11 tabelas no total, projetadas para armazenar e gerenciar informações relacionadas aos usuários de uma rede social, suas interações e atividades. Entre elas, estão tabelas que registram dados essenciais, como perfis de usuários, postagens, comentários e grupos, além de tabelas complementares que gerenciam conexões entre usuários, notificações, avaliações de postagens e comentários, e a organização de tags. Cada tabela desempenha um papel específico no sistema, garantindo um armazenamento

eficiente e permitindo consultas detalhadas sobre os relacionamentos e atividades na plataforma.

2.1 Tabela Usuários

Esta tabela armazena informações sobre os usuários. Os atributos principais incluem:

- id_usuario: Chave primária gerada automaticamente.
- nome_usuário: Nome do Usuário.
- email: Endereço de email.
- data_nascimento: Data de nascimento.
- foto_perfil: Imagem de Perfil.
- data_criacao: Data de Criação do perfil de Usuário.

2.2 Tabela CONEXÕES

A Esta tabela armazena as conexões entre usuários, representando relacionamentos (como amizades ou seguimentos) na rede social. Os atributos principais incluem:

- id_conexao**:** Chave primária gerada automaticamente para identificar cada conexão.
- id_usuario1: Referência ao primeiro usuário envolvido na conexão, vinculada à tabela usuarios por meio de chave estrangeira.
- id_usuario2: Referência ao segundo usuário envolvido na conexão, também vinculada à tabela usuarios.
- data_conexao: Data e hora em que a conexão foi estabelecida, com valor padrão sendo o momento atual.
- UNIQUE (id_usuario1, id_usuario2**)**: Garante que a relação entre dois usuários seja única e não se repita.

2.3 Tabela POSTAGENS

Esta tabela armazena as postagens feitas pelos usuários na rede social. Os atributos principais incluem:

- id_postagem: Chave primária gerada automaticamente para identificar cada postagem.
- id_usuario: Referência ao usuário que criou a postagem, vinculada à tabela usuarios por meio de chave estrangeira.

- data_criacao: Data e hora em que a postagem foi criada, com valor padrão sendo o momento atual.
- conteudo: Texto ou conteúdo da postagem, armazenado como um campo de tipo TEXT.
- tipo: Tipo de conteúdo da postagem, que pode ser texto, imagem ou vídeo, definido por meio de um campo do tipo ENUM.

2.4 Tabela AVALICOES_POSTAGENS

Esta tabela armazena as avaliações feitas pelos usuários nas postagens, como curtidas ou discutidas. Os atributos principais incluem:

- id_avaliacoes: Chave primária gerada automaticamente para identificar cada avaliação.
- id_usuario: Referência ao usuário que fez a avaliação, vinculada à tabela usuarios por meio de chave estrangeira.
- id_postagem: Referência à postagem que foi avaliada, vinculada à tabela postagens por meio de chave estrangeira.
- avaliacao: Tipo de avaliação feita pelo usuário, que pode ser Like (curtir) ou Deslike (discutir), armazenado como um campo ENUM.
- data_avaliacao: Data e hora em que a avaliação foi realizada, com valor padrão sendo o momento atual.
- UNIQUE (id_usuario, id_postagem): Garante que um usuário só possa avaliar uma postagem uma vez, evitando avaliações duplicadas.

2.5 Tabela COMENTARIOS

Esta tabela armazena os comentários feitos pelos usuários nas postagens, incluindo a possibilidade de criar respostas a outros comentários. Os atributos principais incluem:

- id_comentario: Chave primária gerada automaticamente para identificar cada comentário.
- id_usuario: Referência ao usuário que fez o comentário, vinculada à tabela usuarios por meio de chave estrangeira.
- id_postagem: Referência à postagem que recebeu o comentário, vinculada à tabela postagens por meio de chave estrangeira.
- id_comentario_2: Referência a um comentário anterior ao qual o comentário atual está respondendo, podendo ser nulo (caso o comentário não seja uma resposta). Este campo é uma chave estrangeira que faz referência à própria tabela comentários.
- conteudo: Texto do comentário, armazenado como um campo de tipo TEXT.

- data_criacao: Data e hora em que o comentário foi criado, com valor padrão sendo o momento atual.

2.6 Tabela AVALIACOES_COMENTARIOS

Esta tabela armazena as avaliações feitas pelos usuários nos comentários, como curtidas ou descurtidas. Os atributos principais incluem:

- id_avalicao_coment: Chave primária gerada automaticamente para identificar cada avaliação de comentário.
- id_usuario: Referência ao usuário que fez a avaliação, vinculada à tabela usuarios por meio de chave estrangeira.
- id_comentario: Referência ao comentário que foi avaliado, vinculada à tabela comentarios por meio de chave estrangeira.
- avaliacao: Tipo de avaliação feita pelo usuário, que pode ser Like (curtir) ou Deslike (descurtir), armazenado como um campo ENUM.
- data_avaliacao: Data e hora em que a avaliação foi realizada, com valor padrão sendo o momento atual.
- UNIQUE (id_usuario, id_comentario): Garante que um usuário só possa avaliar um comentário uma vez, evitando avaliações duplicadas.

2.7 Tabela NOTIFICACOES

Esta tabela armazena as notificações enviadas aos usuários, informando-os sobre atividades relevantes na rede social, como curtidas ou comentários em suas postagens. Os atributos principais incluem:

- id_notificacao: Chave primária gerada automaticamente para identificar cada notificação.
- id_usuario: Referência ao usuário que recebeu a notificação, vinculada à tabela usuarios por meio de chave estrangeira.
- tipo_acao: Tipo de ação que gerou a notificação, podendo ser Like ou Comentário, armazenado como um campo ENUM.
- id_acao: Identificador da ação que gerou a notificação (por exemplo, o ID de uma postagem ou comentário relacionado à ação).
- origem: Origem da notificação, que pode ser Postagem ou Comentário, indicando se a ação ocorreu em uma postagem ou em um comentário, também definido como um campo ENUM.
- data_notificacao: Data e hora em que a notificação foi gerada, com valor padrão sendo o momento atual.
- lida: Booleano que indica se a notificação foi lida pelo usuário. Seu valor padrão é FALSE, significando que a notificação ainda não foi visualizada.

2.8 Tabela GRUPOS

Esta tabela armazena informações sobre os grupos criados dentro da rede social, que podem ser usados para organizar usuários com interesses comuns. Os atributos principais incluem:

- id_grupo: Chave primária gerada automaticamente para identificar cada grupo.
- nome: Nome do grupo, que deve ser único e não pode ser nulo.
- descricao: Descrição do grupo, fornecendo mais detalhes sobre seu objetivo ou tema.
- data_criacao: Data e hora em que o grupo foi criado, com valor padrão sendo o momento atual.

2.9 Tabela MEMBROS_GRUPO

Esta tabela armazena as informações sobre os usuários que são membros de grupos dentro da rede social, incluindo suas funções dentro do grupo, como membro comum ou administrador. Os atributos principais incluem:

- id_membro_grupo: Chave primária gerada automaticamente para identificar cada membro de um grupo.
- id_grupo: Referência ao grupo ao qual o usuário pertence, vinculada à tabela grupos por meio de chave estrangeira.
- id_usuario: Referência ao usuário que é membro do grupo, vinculada à tabela usuarios por meio de chave estrangeira.
- funcao: Função do usuário dentro do grupo, que pode ser membro (padrão) ou administrador, armazenado como um campo ENUM.
- UNIQUE (id_grupo, id_usuario): Garante que cada usuário só possa ser membro de um grupo uma única vez.

2.10 Tabela TAGS

Esta tabela armazena as tags que podem ser associadas a postagens ou grupos para facilitar a organização e a busca por temas ou tópicos específicos. Os atributos principais incluem:

- id_tag: Chave primária gerada automaticamente para identificar cada tag.
- nome_tag: Nome da tag, que deve ser único e não pode ser nulo.

2.11 Tabela USUARIOS_TAGS

Esta tabela armazena a relação entre os usuários e as tags que eles atribuem a si mesmos, permitindo que os usuários sejam associados a temas ou tópicos específicos. Os atributos principais incluem:

- id_usuario: Referência ao usuário, vinculada à tabela usuarios por meio de chave estrangeira.
- id_tag: Referência à tag atribuída ao usuário, vinculada à tabela tags por meio de chave estrangeira.
- data_atribuicao: Data e hora em que a tag foi atribuída ao usuário, com valor padrão sendo o momento atual.
- PRIMARY KEY (id_usuario, id_tag): Garante que um usuário possa ser associado a uma tag específica apenas uma vez.

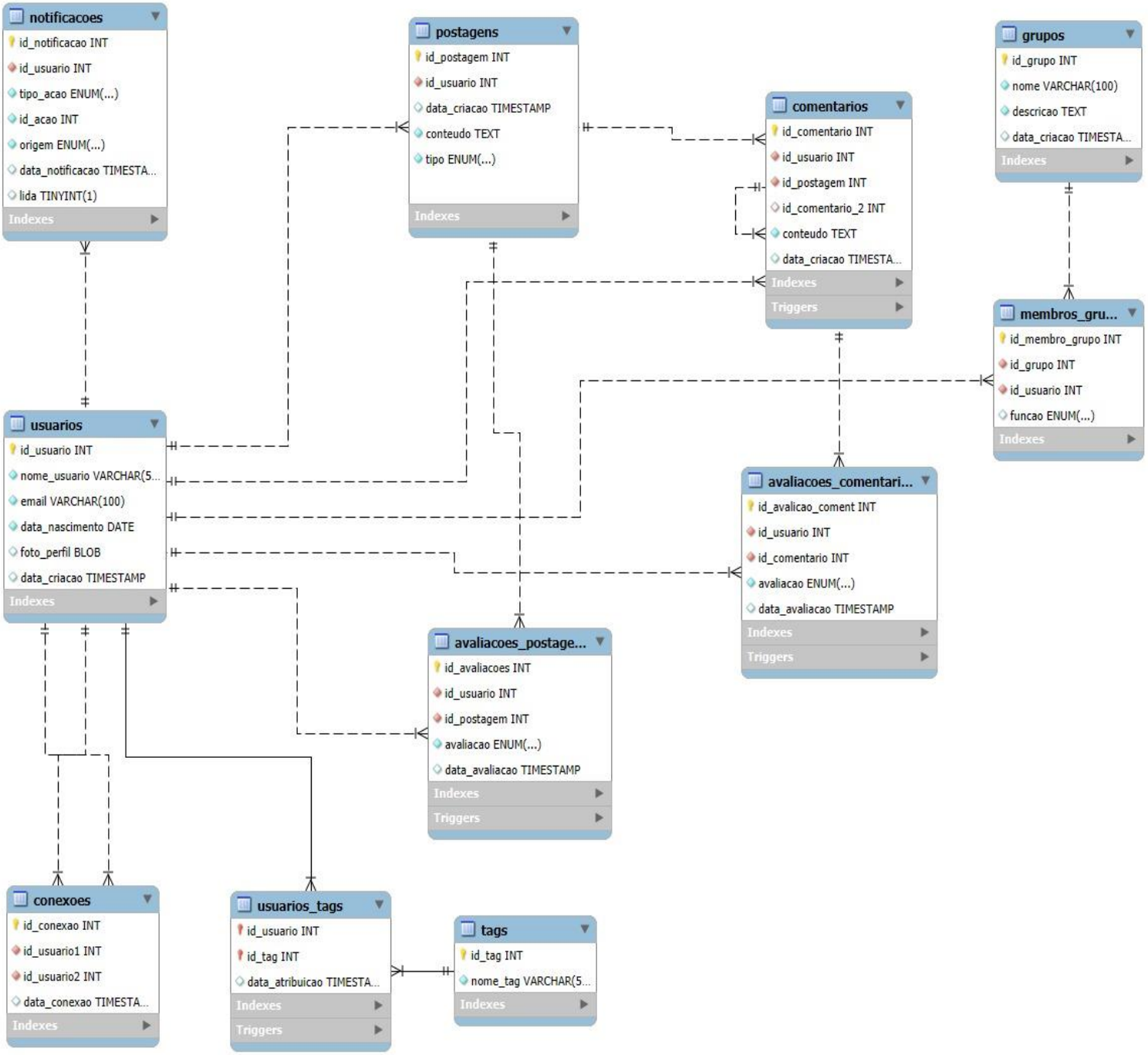
3. Processo de Extração e Importação dos Dados

Os dados foram criados manualmente para simular um ambiente real de rede social. Cada tabela foi projetada com base nas interações típicas dos usuários em plataformas sociais, utilizando boas práticas de modelagem de banco de dados relacional. A inserção dos dados foi realizada por meio de comandos SQL, após a definição da estrutura das tabelas, para garantir que as relações entre usuários, postagens, comentários e outras entidades fossem corretamente estabelecidas e pudessem ser consultadas de forma eficiente.

3.1 Ferramentas Utilizadas

- **MySQL Workbench**: Para modelagem do banco de dados e execução dos scripts SQL.

4. Diagrama Entidade-Relacionamento



Script para criação das tabelas

```sql
CREATE DATABASE IF NOT EXISTS bd_redesocial;
USE bd_redesocial;

CREATE TABLE IF NOT EXISTS usuarios (
    id_usuario INT AUTO_INCREMENT PRIMARY KEY,
    nome_usuario VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    data_nascimento DATE NOT NULL,
    foto_perfil BLOB,
    data_criacao TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS conexoes (
    id_conexao INT AUTO_INCREMENT PRIMARY KEY,
    id_usuario1 INT NOT NULL,
    id_usuario2 INT NOT NULL,
    data_conexao TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (id_usuario1) REFERENCES usuarios(id_usuario),
    FOREIGN KEY (id_usuario2) REFERENCES usuarios(id_usuario),
    UNIQUE (id_usuario1, id_usuario2)
);

CREATE TABLE IF NOT EXISTS postagens (
    id_postagem INT AUTO_INCREMENT PRIMARY KEY,
    id_usuario INT NOT NULL,
    data_criacao TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    conteudo TEXT NOT NULL,
    tipo ENUM('texto', 'imagem', 'video') NOT NULL,
    FOREIGN KEY (id_usuario) REFERENCES usuarios(id_usuario)
);

CREATE TABLE IF NOT EXISTS avaliacoes_postagens (
    id_avaliacoes INT AUTO_INCREMENT PRIMARY KEY,
    id_usuario INT NOT NULL,
    id_postagem INT NOT NULL,
    avaliacao ENUM('Like', 'Deslike') NOT NULL,
    data_avaliacao TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (id_usuario) REFERENCES usuarios(id_usuario),
    FOREIGN KEY (id_postagem) REFERENCES postagens(id_postagem),
    UNIQUE (id_usuario, id_postagem)
);

CREATE TABLE IF NOT EXISTS comentarios (
    id_comentario INT AUTO_INCREMENT PRIMARY KEY,
    id_usuario INT NOT NULL,
```

```
    id_postagem INT NOT NULL,
    id_comentario_2 INT DEFAULT NULL,
    conteudo TEXT NOT NULL,
    data_criacao TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (id_usuario) REFERENCES usuarios(id_usuario),
    FOREIGN KEY (id_postagem) REFERENCES postagens(id_postagem),
    FOREIGN KEY (id_comentario_2) REFERENCES comentarios(id_comentario)
);

CREATE TABLE IF NOT EXISTS avaliacoes_comentarios (
    id_avalicao_coment INT AUTO_INCREMENT PRIMARY KEY,
    id_usuario INT NOT NULL,
    id_comentario INT NOT NULL,
    avaliacao ENUM('Like', 'Deslike') NOT NULL,
    data_avaliacao TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (id_usuario) REFERENCES usuarios(id_usuario),
    FOREIGN KEY (id_comentario) REFERENCES comentarios(id_comentario),
    UNIQUE (id_usuario, id_comentario)
);

CREATE TABLE IF NOT EXISTS notificacoes (
    id_notificacao INT AUTO_INCREMENT PRIMARY KEY,
    id_usuario INT NOT NULL,
    tipo_acao ENUM('Like', 'Comentário') NOT NULL,
    id_acao INT NOT NULL,
    origem ENUM('Postagem', 'Comentário') NOT NULL,
    data_notificacao TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    lida BOOLEAN DEFAULT FALSE,
    FOREIGN KEY (id_usuario) REFERENCES usuarios(id_usuario)
);

CREATE TABLE grupos (
    id_grupo INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) UNIQUE NOT NULL,
    descricao TEXT NOT NULL,
    data_criacao TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE membros_grupo (
    id_membro_grupo INT AUTO_INCREMENT PRIMARY KEY,
    id_grupo INT NOT NULL,
    id_usuario INT NOT NULL,
    funcao ENUM('membro', 'administrador') DEFAULT 'membro',
    FOREIGN KEY (id_grupo) REFERENCES grupos(id_grupo),
    FOREIGN KEY (id_usuario) REFERENCES usuarios(id_usuario),
    UNIQUE (id_grupo, id_usuario)
);

CREATE TABLE IF NOT EXISTS tags (
```

```
    id_tag INT AUTO_INCREMENT PRIMARY KEY,
    nome_tag VARCHAR(50) UNIQUE NOT NULL
);

CREATE TABLE IF NOT EXISTS usuarios_tags (
    id_usuario INT NOT NULL,
    id_tag INT NOT NULL,
    data_atribuicao TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (id_usuario, id_tag),
    FOREIGN KEY (id_usuario) REFERENCES usuarios(id_usuario),
    FOREIGN KEY (id_tag) REFERENCES tags(id_tag)
);
```

Script para inserção dos dados

```
INSERT INTO usuarios (nome_usuario, email, data_nascimento) VALUES
('Ana Silva', 'ana.silva@gmail.com', '1990-04-15'),
('Bruno Costa', 'bruno.costa@yahoo.com', '1985-09-22'),
('Carla Mendes', 'carla.mendes@outlook.com', '1992-06-10'),
('Diego Ferreira', 'diego.ferreira@gmail.com', '1988-03-14'),
('Ester Souza', 'ester.souza@hotmail.com', '1995-12-05'),
('Felipe Lima', 'felipe.lima@gmail.com', '1999-01-20'),
('Gabriela Alves', 'gabriela.alves@uol.com.br', '2001-08-18'),
('Hugo Martins', 'hugo.martins@icloud.com', '1983-11-30'),
('Isabela Rocha', 'isabela.rocha@gmail.com', '1997-07-09'),
('João Pedro', 'joao.pedro@yahoo.com', '1993-02-28'),
('Karina Duarte', 'karina.duarte@live.com', '2000-05-16'),
('Lucas Oliveira', 'lucas.oliveira@gmail.com', '1994-10-02'),
('Mariana Dias', 'mariana.dias@hotmail.com', '1987-11-12'),
('Nicolas Ribeiro', 'nicolas.ribeiro@gmail.com', '2002-09-25'),
('Olivia Castro', 'olivia.castro@uol.com.br', '1998-04-08');

INSERT INTO conexoes (id_usuario1, id_usuario2) VALUES
(1, 2), (2, 3), (3, 4), (4, 5), (5, 6),
(6, 7), (7, 8), (8, 9), (9, 10), (10, 11),
(11, 12), (12, 13), (13, 14), (14, 15), (15, 1);

INSERT INTO conexoes (id_usuario1, id_usuario2) VALUES
(1, 3), (2, 4), (3, 5), (4, 6), (5, 7),
(6, 8), (7, 9), (8, 10), (9, 11), (10, 12),
(11, 13), (12, 14), (13, 15), (14, 1), (15, 2);

INSERT INTO conexoes (id_usuario1, id_usuario2) VALUES
(1, 4), (2, 5), (3, 6), (4, 7), (5, 8),
(6, 9), (7, 10), (8, 11), (9, 12), (10, 13),
```

```
(11, 14), (12, 15), (13, 1), (14, 2), (15, 3);

-- Inserindo 15 postagens
INSERT INTO postagens (id_usuario, conteudo, tipo) VALUES
(1, 'Primeira postagem de Ana!', 'texto'),
(2, 'Bruno compartilhou uma foto!', 'imagem'),
(3, 'Pensando sobre a vida hoje.', 'texto'),
(4, 'Diego está curtindo a praia!', 'video'),
(5, 'Uma imagem para alegrar o dia.', 'imagem'),
(6, 'Felipe lançou um pensamento profundo.', 'texto'),
(7, 'Gabriela compartilhou um vídeo divertido.', 'video'),
(8, 'Hugo postou um artigo interessante.', 'texto'),
(9, 'Isabela mostrando sua nova arte.', 'imagem'),
(10, 'João postou uma receita deliciosa!', 'texto'),
(11, 'Karina compartilhou um clipe musical.', 'video'),
(12, 'Lucas divulgou seu novo projeto.', 'imagem'),
(13, 'Mariana compartilhou um texto motivador.', 'texto'),
(14, 'Nicolas mostrando sua nova tatuagem.', 'imagem'),
(15, 'Olivia falando sobre viagens.', 'texto');

INSERT INTO avaliacoes_postagens (id_usuario, id_postagem, avaliacao)
VALUES
(1, 2, 'Like'), (2, 3, 'Like'), (3, 4, 'Deslike'),
(4, 5, 'Like'), (5, 6, 'Deslike'), (6, 7, 'Like'),
(7, 8, 'Like'), (8, 9, 'Deslike'), (9, 10, 'Like'),
(10, 11, 'Deslike'), (11, 12, 'Like'), (12, 13, 'Like'),
(13, 14, 'Deslike'), (14, 15, 'Like'), (15, 1, 'Like');

-- Inserindo 15 comentários
INSERT INTO comentarios (id_usuario, id_postagem, conteudo) VALUES
(1, 2, 'Muito interessante!'),
(2, 3, 'Adorei sua reflexão.'),
(3, 4, 'Que lugar incrível!'),
(4, 5, 'Foto maravilhosa!'),
(5, 6, 'Você tem razão!'),
(6, 7, 'Hahaha, que divertido!'),
(7, 8, 'Gostei do artigo.'),
(8, 9, 'Sua arte é incrível!'),
(9, 10, 'Vou tentar essa receita.'),
(10, 11, 'Que música legal!'),
(11, 12, 'Ótimo projeto!'),
(12, 13, 'Inspirador!'),
(13, 14, 'Ficou incrível!'),
(14, 15, 'Adoro viajar também.'),
(15, 1, 'Parabéns pelo conteúdo!');

INSERT INTO avaliacoes_comentarios (id_usuario, id_comentario, avaliacao)
VALUES
(2, 1, 'Like'), (3, 2, 'Like'), (4, 3, 'Deslike'),
```

```
(5, 4, 'Like'), (6, 5, 'Like'), (7, 6, 'Deslike'),
(8, 7, 'Like'), (9, 8, 'Like'), (10, 9, 'Deslike'),
(11, 10, 'Like'), (12, 11, 'Like'), (13, 12, 'Deslike'),
(14, 13, 'Like'), (15, 14, 'Like'), (1, 15, 'Deslike');

-- Inserindo 15 notificações
INSERT INTO notificacoes (id_usuario, tipo_acao, id_acao, origem, lida)
VALUES
(1, 'Like', 2, 'Postagem', FALSE),
(2, 'Comentário', 3, 'Postagem', TRUE),
(3, 'Like', 4, 'Comentário', FALSE),
(4, 'Like', 5, 'Postagem', TRUE),
(5, 'Comentário', 6, 'Comentário', FALSE),
(6, 'Like', 7, 'Postagem', TRUE),
(7, 'Like', 8, 'Comentário', FALSE),
(8, 'Comentário', 9, 'Postagem', TRUE),
(9, 'Like', 10, 'Comentário', FALSE),
(10, 'Like', 11, 'Postagem', TRUE),
(11, 'Comentário', 12, 'Postagem', FALSE),
(12, 'Like', 13, 'Comentário', TRUE),
(13, 'Like', 14, 'Postagem', FALSE),
(14, 'Comentário', 15, 'Comentário', TRUE),
(15, 'Like', 1, 'Postagem', FALSE);

INSERT INTO grupos (nome, descricao) VALUES
('Grupo de Música', 'Discussões sobre música.'),
('Grupo de Tecnologia', 'Novidades e tendências.'),
('Grupo de Viagem', 'Dicas e experiências de viagem.'),
('Grupo de Fotografia', 'Compartilhe suas fotos.'),
('Grupo de Livros', 'Recomendações literárias.');

-- Inserindo 15 membros em grupos
INSERT INTO membros_grupo (id_grupo, id_usuario, funcao) VALUES
(1, 1, 'administrador'), (1, 2, 'membro'),
(2, 3, 'administrador'), (2, 4, 'membro'),
(3, 5, 'administrador'), (3, 6, 'membro'),
(4, 7, 'administrador'), (4, 8, 'membro'),
(5, 9, 'administrador'), (5, 10, 'membro'),
(1, 11, 'membro'), (2, 12, 'membro'),
(3, 13, 'membro'), (4, 14, 'membro'),
(5, 15, 'membro');

INSERT INTO tags (nome_tag) VALUES
('Música'), ('Tecnologia'), ('Viagem'),
('Fotografia'), ('Livros'), ('Culinária'),
('Esportes'), ('Moda'), ('Cinema'),
('Arte'), ('Jogos'), ('Fitness'),
('Educação'), ('Saúde'), ('Negócios');
```

```sql
INSERT INTO usuarios_tags (id_usuario, id_tag) VALUES
(1, 1), (1, 2), (1, 3), (2, 4), (2, 5),
(3, 6), (3, 7), (4, 8), (4, 9), (5, 10),
(5, 11), (6, 12), (6, 13), (7, 14), (7, 15);

INSERT INTO comentarios (id_usuario, id_postagem, conteudo) VALUES
(2, 1, 'Muito bem colocado, Ana!'),
(3, 1, 'Concordo totalmente!'),
(4, 2, 'Bruno, essa foto está incrível!'),
(5, 2, 'Que lugar é esse? Parece lindo!'),
(6, 3, 'Você sempre com ótimas reflexões, Mariana.'),
(7, 4, 'Queria estar aí também, Diego!'),
(8, 5, 'Que talento! Parabéns.'),
(9, 6, 'Adorei esse pensamento. Muito profundo!'),
(10, 7, 'Esse vídeo é hilário!'),
(11, 8, 'Muito informativo, Hugo.'),
(12, 9, 'Que obra incrível, Isabela!'),
(13, 10, 'Vou testar essa receita no fim de semana.'),
(14, 11, 'Essa música me trouxe boas lembranças!'),
(15, 12, 'Ótima iniciativa, Lucas.'),
(1, 15, 'Adoro viajar também!'),
(3, 15, 'Quais foram os destinos que você mais gostou?');

INSERT INTO avaliacoes_postagens (id_usuario, id_postagem, avaliacao)
VALUES
(1, 1, 'Like'),
(2, 2, 'Like'),
(3, 3, 'Like'),
(4, 4, 'Like'),
(5, 5, 'Like'),
(6, 6, 'Like'),
(7, 7, 'Like'),
(8, 8, 'Like'),
(9, 9, 'Like'),
(10, 10, 'Like'),
(11, 11, 'Like'),
(12, 12, 'Like'),
(13, 13, 'Like'),
(14, 14, 'Like'),
(15, 15, 'Like'),
(3, 2, 'Like'),
(8, 3, 'Like'),
(5, 4, 'Like'),
(7, 5, 'Like'),
(6, 9, 'Like');

INSERT INTO avaliacoes_comentarios (id_usuario, id_comentario, avaliacao)
VALUES
(1, 1, 'Like'),
```

```
(2, 2, 'Like'),
(3, 3, 'Like'),
(4, 4, 'Like'),
(5, 5, 'Like'),
(6, 6, 'Like'),
(7, 7, 'Like'),
(8, 8, 'Like'),
(9, 9, 'Like'),
(10, 10, 'Like'),
(11, 11, 'Like'),
(12, 12, 'Like'),
(13, 13, 'Like'),
(14, 14, 'Like'),
(15, 15, 'Like'),
(1, 3, 'Like'),
(2, 4, 'Like'),
(3, 5, 'Like');
```

Script para Procedures e Triggers

```
USE bd_redesocial;

DELIMITER $$
CREATE TRIGGER trg_limite_tags_insert
BEFORE INSERT ON usuarios_tags
FOR EACH ROW
BEGIN
    DECLARE total_tags INT;

    SELECT COUNT(*) INTO total_tags
    FROM usuarios_tags
    WHERE id_usuario = NEW.id_usuario;

    IF total_tags >= 5 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'O limite de 5 tags por usuário foi
atingido.';
    END IF;
END$$

CREATE TRIGGER trg_limite_tags_update
BEFORE UPDATE ON usuarios_tags
FOR EACH ROW
BEGIN
    DECLARE total_tags INT;
```

```
    SELECT COUNT(*) INTO total_tags
    FROM usuarios_tags
    WHERE id_usuario = NEW.id_usuario AND id_tag != OLD.id_tag;

    IF total_tags >= 5 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'O limite de 5 tags por usuário foi
atingido.';
    END IF;
END$$
DELIMITER ;

DELIMITER $$
CREATE PROCEDURE registrar_notificacao(
    IN p_id_usuario INT,
    IN p_tipo_acao ENUM('Like', 'Comentário'),
    IN p_id_acao INT,
    IN p_origem ENUM('Postagem', 'Comentário')
)
BEGIN
    INSERT INTO notificacoes (id_usuario, tipo_acao, id_acao, origem,
data_notificacao)
    VALUES (p_id_usuario, p_tipo_acao, p_id_acao, p_origem,
CURRENT_TIMESTAMP);
END $$
DELIMITER ;

DELIMITER $$
CREATE TRIGGER trigger_like_postagem
AFTER INSERT ON avaliacoes_postagens
FOR EACH ROW
BEGIN
    CALL registrar_notificacao(NEW.id_usuario, 'Like', NEW.id_postagem,
'Postagem');
END $$
DELIMITER ;

DELIMITER $$
CREATE TRIGGER trigger_like_comentario
AFTER INSERT ON avaliacoes_comentarios
FOR EACH ROW
BEGIN
    CALL registrar_notificacao(NEW.id_usuario, 'Like', NEW.id_comentario,
'Comentário');
END $$
DELIMITER ;

DELIMITER $$
```

```
CREATE TRIGGER trigger_comentario_postagem
AFTER INSERT ON comentarios
FOR EACH ROW
BEGIN
    CALL registrar_notificacao(NEW.id_usuario, 'Comentário',
NEW.id_postagem, 'Postagem');
END $$
DELIMITER ;

DELIMITER $$
CREATE TRIGGER trigger_comentario_comentario
AFTER INSERT ON comentarios
FOR EACH ROW
BEGIN
    CALL registrar_notificacao(NEW.id_usuario, 'Comentário',
NEW.id_comentario_2, 'Comentário');
END $$
DELIMITER ;
```

5. Consultas Realizadas

Primeira consulta:

```
SELECT
    a.id_postagem,
    a.conteudo,
    b.total_comentarios,
    c.total_avaliacoes
FROM
    postagens a
left join (
    SELECT
        xa.id_postagem,
        COUNT(*) as total_comentarios
    FROM
        comentarios xa
    group by
        xa.id_postagem
) b on a.id_postagem = b.id_postagem
left join (
    SELECT
        xa.id_postagem,
        COUNT(*) as total_avaliacoes
    FROM
```

```
        avaliacoes_postagens xa
    GROUP BY
        xa.id_postagem
) c on a.id_postagem = c.id_postagem
ORDER BY
    b.total_comentarios + c.total_avaliacoes DESC;
```

Segunda consulta:

```
SELECT
    A.id_usuario,
    A.nome_usuario,
    COUNT(DISTINCT C.id_avaliacoes) AS total_curtidas,
    COUNT(DISTINCT D.id_comentario) AS total_comentarios,
    (COUNT(DISTINCT C.id_avaliacoes) + COUNT(DISTINCT D.id_comentario))
AS total_engajamento
FROM
    usuarios A
LEFT JOIN postagens B ON
    A.id_usuario = B.id_usuario
LEFT JOIN avaliacoes_postagens C ON
    B.id_postagem = C.id_postagem
LEFT JOIN comentarios D ON
    B.id_postagem = D.id_postagem
GROUP BY
    A.id_usuario
ORDER BY
    total_engajamento DESC,
    A.nome_usuario;
```

Terceira consulta:

```
SELECT
    HOUR(data_criacao) AS hora,
    COUNT(*) AS total_atividades
FROM (
    SELECT
        data_criacao
    FROM postagens
    UNION ALL
    SELECT
        data_avaliacao
    FROM
        avaliacoes_postagens
    UNION ALL
```

```
    SELECT
        data_criacao
    FROM
        comentarios
) atividades
GROUP BY
    hora
ORDER BY
    total_atividades DESC;
```

Quarta consulta:

```
SELECT
    a.nome AS nome_grupo,
    COUNT(DISTINCT c.id_tag) AS total_tags_diversas
FROM
    grupos a
INNER JOIN membros_grupo b ON
    a.id_grupo = b.id_grupo
INNER JOIN usuarios_tags c ON
    b.id_usuario = c.id_usuario
GROUP BY
    a.id_grupo
ORDER BY
    total_tags_diversas DESC;
```

Quinta Consulta:

```
SELECT
    a.id_usuario,
    a.nome_usuario,
    COUNT(CASE WHEN c.avaliacao = 'Deslike' THEN 1 END) AS
total_deslikes,
    COUNT(c.id_avaliacoes) AS total_avaliacoes,
    ROUND((COUNT(CASE WHEN c.avaliacao = 'Deslike' THEN 1 END) /
COUNT(c.id_avaliacoes)) * 100, 2) AS proporcao_deslikes
FROM
    usuarios A
LEFT JOIN postagens b ON
    a.id_usuario = b.id_usuario
LEFT JOIN avaliacoes_postagens c ON
    b.id_postagem = c.id_postagem
GROUP BY
    a.id_usuario
HAVING
```

```
    total_avaliacoes > 0
ORDER BY
    proporcao_deslikes DESC,
    total_deslikes DESC;
```

Explicação das consultas:

Query 1: Postagens com total de comentários e avaliações

Objetivo: Esta consulta retorna uma lista de postagens, junto com o conteúdo, o número total de comentários e o número total de avaliações (curtidas/descurtidas) em cada postagem.
Explicação:

- A tabela postagens (aliás a) é unida com duas subconsultas:
  - **Subconsulta b**: Conta o número total de comentários (total_comentarios) para cada postagem.
  - **Subconsulta c**: Conta o número total de avaliações (total_avaliacoes) para cada postagem.
- A junção é feita usando o LEFT JOIN, garantindo que todas as postagens apareçam, mesmo que não tenham comentários ou avaliações.
- O resultado é ordenado por uma soma de total_comentarios e total_avaliacoes, em ordem decrescente, ou seja, as postagens com mais interações (comentários e avaliações) aparecem primeiro.

Query 2: Total de curtidas, comentários e engajamento por usuário

Objetivo: Esta consulta calcula o total de curtidas (avaliações), comentários e o engajamento geral por usuário.
Explicação:

- A tabela usuarios (aliás A) é unida com as tabelas postagens (B), avaliacoes_postagens (C) e comentarios (D) para contar as curtidas e comentários de cada usuário.
- **COUNT(DISTINCT C.id_avaliacoes)** conta o número de curtidas únicas feitas pelo usuário.
- **COUNT(DISTINCT D.id_comentario)** conta o número de comentários únicos feitos pelo usuário.
- A soma desses dois valores é chamada de total_engajamento, representando o total de interações (curtidas + comentários) por usuário.
- O resultado é agrupado por id_usuario e ordenado primeiro pelo engajamento total (de forma decrescente), e depois pelo nome do usuário.

Query 3: Atividades por hora

Objetivo: Esta consulta mostra o número total de atividades (postagens, avaliações e comentários) realizadas por hora.
Explicação:

- A consulta usa o operador UNION ALL para combinar as datas de criação de postagens, avaliações e comentários em uma única lista de atividades.
- O HOUR(data_criacao) extrai a hora da data de criação de cada atividade.
- As atividades são agrupadas por hora, e o número total de atividades realizadas em cada hora é contado.
- O resultado é ordenado pelo total de atividades de forma decrescente, mostrando as horas com mais atividades primeiro.

Query 4: Grupos com o maior número de tags distintas

Objetivo: Esta consulta retorna os grupos com o maior número de tags distintas associadas aos membros.
Explicação:

- A tabela grupos (aliás a) é unida com a tabela membros_grupo (b), que lista os membros de cada grupo, e com a tabela usuarios_tags (c), que lista as tags associadas aos usuários.
- A contagem de tags distintas associadas a cada grupo é feita com o COUNT(DISTINCT c.id_tag).
- O resultado é agrupado por grupo (a.id_grupo) e ordenado pelo total de tags distintas, em ordem decrescente.

Query 5: Proporção de descurtidas por usuário

Objetivo: Esta consulta calcula a proporção de descurtidas (Deslike) feitas por cada usuário em relação ao total de avaliações feitas. Explicação:

- A tabela usuarios (aliás A) é unida com postagens (b) e avaliacoes_postagens (c) para obter as avaliações feitas por cada usuário.
- COUNT(CASE WHEN c.avaliacao = 'Deslike' THEN 1 END) conta o número de descurtidas feitas pelo usuário.
- COUNT(c.id_avaliacoes) conta o total de avaliações feitas pelo usuário, seja "curtidas" ou "descurtidas".
- A proporção de descurtidas é calculada dividindo o número de descurtidas pelo total de avaliações e multiplicando por 100.
- A consulta é agrupada por id_usuario e ordenada primeiro pela proporção de descurtidas (em ordem decrescente) e depois pelo número total de descurtidas.

Conclusão

O desenvolvimento deste banco de dados para rede social proporcionou uma solução eficiente para o gerenciamento e consulta de informações

detalhadas sobre usuários, postagens, interações e grupos. A modelagem conceitual e implementação em MySQL permitiram a criação de um sistema organizado que facilita o acesso e a manipulação dos dados, atendendo aos requisitos do projeto.

Com a criação das principais tabelas, **USUÁRIOS**, **POSTAGENS**, **COMENTÁRIOS**, **AVALIAÇÕES**, **GRUPOS**, **MEMBROS_GRUPO** e **TAGS**, foi possível representar de forma abrangente os aspectos essenciais de uma rede social. A utilização de chaves estrangeiras garantiu a integridade referencial entre as tabelas, enquanto as regras de validação asseguraram a consistência dos dados inseridos, promovendo uma experiência de interação eficiente entre os usuários e o conteúdo da plataforma.

Referências:

W3 Schools