



TECNÓLOGO EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
TRABALHO DE BANCO DE DADOS

CAIO ALVES NASCIMENTO
LÁZARO PEDRO MARTINS SANTOS
MILENA PIRES MENDES

**DESENVOLVIMENTO DE UM BANCO DE DADOS PARA UM
SISTEMA DE GESTÃO DE E-COMMERCE DE LIVROS**

GUANAMBI - BA

2025

CAIO ALVES NASCIMENTO
LÁZARO PEDRO MARTINS SANTOS
MILENA PIRES MENDES

DESENVOLVIMENTO DE UM BANCO DE DADOS PARA UM SISTEMA DE GESTÃO DE E-COMMERCE DE LIVROS

Relatório apresentado como requisito para aprovação na disciplina de **Tópicos avançados em banco de dados** do curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia Baiano - *Campus* Guanambi.

Orientador(a): Carlos Anderson Oliveira Silva.

GUANAMBI - BA

2025

SUMÁRIO

1. INTRODUÇÃO.....	3
2. MODELAGEM DO BANCO DE DADOS.....	4
2.1. TABELAS, CHAVES PRIMÁRIAS E ESTRANGEIRAS.....	4
2.2. ESTRUTURA SQL DE CADA TABELA E RESTRIÇÕES APLICADAS.....	5
3. POVOAMENTO DE DADOS (DML).....	6
3.1. POVOAMENTO DAS TABELAS.....	6
4. VIEWS.....	8
4.1. VW_LIVROS MAIS VENDIDOS.....	8
4.2. VW_CLIENTES_ATIVOS.....	8
4.3. VW_ESTOQUE_BAIXO.....	9
5. STORED PROCEDURES.....	10
5.1. SP_INSERTIR_LIVRO.....	10
5.2. SP_ATUALIZAR_ESTOQUE.....	11
5.3. SP_GERAR_RELATORIO_VENDAS (Relatório de Vendas por Categoria).....	12
5.4. SP_RELATORIO_VENDAS_POR_AUTOR (Relatório de Vendas por Autor).....	12
5.5. SP_RELATORIO_VENDAS_POR_PERIODO (Relatório de Vendas por Período Geral).....	13
5.6. SP_CADASTRAR_CLIENTE.....	13
5.7. SP_CRIAR_PEDIDO (Criação de Pedido com Transação).....	14
6. STORED FUNCTION.....	16
6.1. FN_CALCULAR_DESCONTO.....	16
7. ÍNDICES E OTIMIZAÇÃO.....	17
8. CONTROLE DE ACESSO E SEGURANÇA.....	18
8.1. CRIAÇÃO DE USUÁRIOS.....	19
8.2. DETALHES DOS USUÁRIOS:.....	19
8.2.1. usuario_cliente.....	19
8.2.2. usuario_tecnico.....	20
8.2.3. usuario_gerente.....	20
9. VALIDAÇÃO DE SEGURANÇA DE SENHAS.....	20
10. CONEXÃO EM AMBIENTE COLABORATIVO (HOST/CLIENTE).....	20
11. POLÍTICA DE BACKUP.....	21
12. CONSULTAS DEMONSTRATIVAS.....	23
13. TESTES E RESULTADOS.....	23
14. CONCLUSÃO.....	23

1. INTRODUÇÃO

Este projeto tem como objetivo o desenvolvimento completo de um banco de dados relacional para a simulação de um sistema de e-commerce de livros. A proposta central consiste em aplicar os conhecimentos teóricos de SQL em um cenário prático e realista, abordando desde a modelagem das entidades até a implementação de funcionalidades avançadas, como views, procedures, functions, controle de acesso e políticas de backup.

Durante o desenvolvimento, buscou-se atender às seguintes diretrizes pedagógicas, estruturação de tabelas com integridade referencial, criação de rotinas automatizadas para inserção, atualização e consulta de dados, otimização de desempenho por meio de índices, garantia de segurança dos dados com criptografia de senhas e controle de usuários, implantação de uma política de backup automatizado com retenção de cópias e recuperação por log binário.

O sistema contempla tabelas para gerenciamento de clientes, livros, pedidos, estoque e categorias, além de recursos para geração de relatórios e controle transacional. Foram também implementados scripts auxiliares para facilitar a configuração, replicação remota e agendamento de backups periódicos. Este trabalho consolida o conhecimento adquirido ao longo da disciplina e demonstra, na prática, como soluções completas podem ser implementadas em um SGBD real para aplicações comerciais.

2. MODELAGEM DO BANCO DE DADOS

Este documento descreve a estrutura do banco de dados `ecommerce_livros_db`, projetado para gerenciar um sistema de vendas online de livros. A modelagem inclui tabelas para categorias, autores, editoras, livros, clientes, pedidos, itens de pedido e carrinho de compras.

2.1. TABELAS, CHAVES PRIMÁRIAS E ESTRANGEIRAS

O banco de dados é composto por oito tabelas principais, cada uma com uma chave primária (**PRIMARY KEY**) para identificação única de seus registros. As relações entre as tabelas são estabelecidas através de chaves estrangeiras (**FOREIGN KEY**), garantindo a integridade referencial.

- **Categorias:** Armazena as diferentes categorias de livros.
 - `id` (PK)
- **Autores:** Contém informações sobre os autores dos livros.
 - `id` (PK)
- **Editoras:** Armazena dados das editoras.
 - `id` (PK)
- **Livros:** O core do catálogo, detalhando cada livro.
 - `id` (PK)
 - `autor_id` (FK para `Autores.id`)
 - `editora_id` (FK para `Editoras.id`)
 - `categoria_id` (FK para `Categorias.id`)
- **Clientes:** Gerencia os usuários do sistema.
 - `id` (PK)
- **Pedidos:** Registra as transações de compra.
 - `id` (PK)
 - `cliente_id` (FK para `Clientes.id`)

- **Itens_Pedido:** Detalha quais livros estão em cada pedido.
 - `id` (PK)
 - `pedido_id` (FK para `Pedidos.id`)
 - `livro_id` (FK para `Livros.id`)
- **Carrinho:** Armazena os itens que um cliente tem no carrinho de compras antes de finalizar um pedido.
 - `id` (PK)
 - `cliente_id` (FK para `Clientes.id`)
 - `livro_id` (FK para `Livros.id`)
 -

2.2. ESTRUTURA SQL DE CADA TABELA E RESTRIÇÕES APLICADAS

```
CREATE TABLE Categorias (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nome VARCHAR(255) NOT NULL UNIQUE
);
```

```
CREATE TABLE Autores (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nome VARCHAR(255) NOT NULL UNIQUE,
  nacionalidade VARCHAR(255),
  data_nascimento DATE
);
```

```
CREATE TABLE Editoras (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nome VARCHAR(255) NOT NULL,
  pais VARCHAR(255)
);
```

```
CREATE TABLE Livros (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  titulo VARCHAR(255) NOT NULL,
  autor_id INT UNSIGNED NOT NULL,
  isbn CHAR(20) UNIQUE NOT NULL,
  editora_id INT UNSIGNED NOT NULL,
  preco DECIMAL(10,2) NOT NULL,
  quantidade_estoque INT NOT NULL,
  categoria_id INT UNSIGNED NOT NULL,
  FOREIGN KEY (autor_id) REFERENCES Autores(id),
  FOREIGN KEY (editora_id) REFERENCES Editoras(id),
  FOREIGN KEY (categoria_id) REFERENCES Categorias(id)
);
```

```

CREATE TABLE Clientes (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nome VARCHAR(255) NOT NULL,
  email VARCHAR(255) UNIQUE NOT NULL,
  hash_senha VARCHAR(255) NOT NULL,
  salt_senha VARCHAR(255) NOT NULL,
  endereco VARCHAR(255) NOT NULL
);

CREATE TABLE Pedidos (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  cliente_id INT UNSIGNED NOT NULL,
  data_pedido DATETIME DEFAULT CURRENT_TIMESTAMP,
  status ENUM('Aberto', 'Enviado', 'Entregue', 'Cancelado') NOT NULL DEFAULT 'Aberto',
  FOREIGN KEY (cliente_id) REFERENCES Clientes(id)
);

CREATE TABLE Itens_Pedido (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  pedido_id INT UNSIGNED NOT NULL,
  livro_id INT UNSIGNED NOT NULL,
  quantidade INT UNSIGNED NOT NULL CHECK (quantidade > 0),
  preco_unitario DECIMAL(10,2) NOT NULL,
  FOREIGN KEY (pedido_id) REFERENCES Pedidos(id),
  FOREIGN KEY (livro_id) REFERENCES Livros(id)
);

CREATE TABLE Carrinho (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  cliente_id INT UNSIGNED NOT NULL,
  livro_id INT UNSIGNED NOT NULL,
  quantidade INT UNSIGNED NOT NULL,
  FOREIGN KEY (cliente_id) REFERENCES Clientes(id),
  FOREIGN KEY (livro_id) REFERENCES Livros(id)
);

```

3. POVOAMENTO DE DADOS (DML)

3.1. POVOAMENTO DAS TABELAS

```

INSERT INTO Categorias (nome) VALUES
('Romance'), ('Terror'), ('Suspense'), ('Ficção Científica'), ('Fantasia'), ('Aventura'), ('Mangá');

```

```

INSERT INTO Editoras (nome, pais) VALUES
('Companhia das Letras', 'Brasil'), ('Penguin Random House', 'EUA'), ('HarperCollins', 'Reino Unido'),
('Planeta', 'Espanha'), ('DarkSide Books', 'Brasil'), ('Editora JBC', 'Brasil'), ('Abril Educação', 'Brasil');

```

```

INSERT INTO Autores (nome, nacionalidade, data_nascimento) VALUES

```

('George R. R. Martin', 'Americana', '1948-09-20'), ('Julia Quinn', 'Americana', '1970-01-12'), ('Masashi Kishimoto', 'Japonesa', '1974-11-08'), ('Stephen King', 'Americana', '1947-09-21'), ('J.K. Rowling', 'Britânica', '1965-07-31'), ('Isaac Asimov', 'Russiana', '1920-01-02'), ('Brandon Sanderson', 'Americana', '1975-12-19'), ('Agatha Christie', 'Britânica', '1890-09-15'), ('Haruki Murakami', 'Japonesa', '1949-01-12'), ('Jules Verne', 'Francesa', '1828-02-08'), ('Paulo Coelho', 'Brasileira', '1947-08-24'), ('Franz Kafka', 'Austriaca', '1883-07-03'), ('Miguel de Cervantes', 'Espanhola', '1547-09-29');

```
INSERT INTO Livros (titulo, autor_id, ISBN, editora_id, preco, quantidade_estoque, categoria_id) VALUES
('A Guerra dos Tronos', 1, '978-8576573980', 4, 69.90, 50, 5),
('A Fúria dos Reis', 1, '978-8576573997', 4, 69.90, 50, 5),
('A Tormenta de Espadas', 1, '978-8576574003', 4, 69.90, 40, 5),
('O Festim dos Corvos', 1, '978-8576574010', 4, 69.90, 35, 5),
('A Dança dos Dragões', 1, '978-8576574027', 4, 69.90, 30, 5),
('O Duque e Eu', 2, '978-8551000001', 1, 39.90, 40, 1),
('O Visconde que me Amava', 2, '978-8551000002', 1, 39.90, 38, 1),
('Naruto Vol. 1', 3, '978-8551001001', 6, 25.00, 60, 7),
('It - A Coisa', 4, '978-8532527365', 1, 49.90, 40, 2),
('Harry Potter e a Pedra Filosofal', 5, '978-8532530785', 1, 59.90, 80, 5),
('Fundação', 6, '978-8576570316', 4, 42.50, 35, 4),
('Assassinato no Expresso Oriente', 8, '978-8525430302', 1, 36.90, 40, 3);
```

```
INSERT INTO Clientes (nome, email, hash_senha, salt_senha, endereco) VALUES
('Milena Pires', 'milenapires94@gmail.com', 'hash_exemplo_1', 'salt_exemplo_1', 'Rua das Flores, 123'),
('Caio Alves', 'caioalves22@gmail.com', 'hash_exemplo_2', 'salt_exemplo_2', 'Av. Brasil, 456'),
('Lázaro Pedro', 'lazaropedro76@gmail.com', 'hash_exemplo_3', 'salt_exemplo_3', 'Rua das Acácias, 789'),
('Carlos Anderson', 'carlosanderson10@gmail.com', 'hash_exemplo_4', 'salt_exemplo_4', 'Av. das Palmeiras, 101'),
('João Miguel', 'joaomiguel58@gmail.com', 'hash_exemplo_5', 'salt_exemplo_5', 'Rua do Sol, 202');
```

```
INSERT INTO Pedidos (cliente_id, data_pedido, status) VALUES
(1, '2025-05-10 10:30:00', 'Entregue'),
(2, '2025-05-12 14:00:00', 'Enviado'),
(3, '2025-05-20 09:15:00', 'Aberto'),
(1, '2025-06-01 18:45:00', 'Aberto');
```

```
INSERT INTO Itens_Pedido (pedido_id, livro_id, quantidade, preco_unitario) VALUES
(1, 1, 1, 69.90), (1, 6, 2, 39.90),
(2, 7, 1, 39.90), (2, 2, 1, 69.90),
(3, 4, 1, 69.90), (3, 9, 1, 49.90),
(4, 10, 2, 59.90);
```

```
INSERT INTO Carrinho (cliente_id, livro_id, quantidade) VALUES
(1, 3, 1), (1, 5, 2),
(2, 8, 1), (2, 11, 1),
(3, 12, 1);
```


4. VIEWS

4.1. VW_LIVROS MAIS VENDIDOS

```
CREATE OR REPLACE VIEW vw_livros_mais_vendidos AS
SELECT
    l.id AS livro_id,
    l.titulo,
    SUM(ip.quantidade) AS total_vendido
FROM Itens_Pedido ip
JOIN Livros l ON ip.livro_id = l.id
JOIN Pedidos p ON ip.pedido_id = p.id
WHERE p.data_pedido >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH)
GROUP BY l.id, l.titulo
ORDER BY total_vendido DESC;
```

Propósito: Está view foi projetada para mostrar quais **livros estão sendo mais vendidos** nos últimos seis meses. É extremamente útil para análises de vendas, para informar decisões de compra de estoque e para destacar os best-sellers no seu site de e-commerce.

Como Funciona:

- Ela **une (JOIN)** as tabelas **Itens_Pedido** (que liga pedidos a livros), **Livros** (para detalhes do livro, como o título) e **Pedidos** (para obter a data do pedido).
- A cláusula **WHERE** filtra apenas os pedidos que ocorreram **nos últimos seis meses** a partir da data atual (**CURDATE()**).
- Em seguida, ela **agrupa (GROUP BY)** os resultados por cada **livro_id** e **título** únicos, permitindo que a função **SUM(ip.quantidade)** calcule a **quantidade total vendida** de cada livro.
- Por fim, ela **ordena (ORDER BY)** esses livros do maior **total_vendido** para o menor.

4.2. VW_CLIENTES_ATIVOS

```
CREATE OR REPLACE VIEW vw_clientes_ativos AS
SELECT DISTINCT
    c.id AS cliente_id, c.nome, c.email,
    MAX(p.data_pedido) AS ultima_compra
FROM Clientes c
JOIN Pedidos p ON p.cliente_id = c.id
WHERE p.data_pedido >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH)
GROUP BY c.id, c.nome, c.email;
```

Propósito: Está view ajuda a identificar seus **clientes ativos** – aqueles que fizeram pelo menos uma compra nos últimos seis meses. Essa informação é vital para campanhas de marketing (por exemplo, segmentar usuários engajados), gestão de relacionamento com o cliente e para entender o engajamento do cliente.

Como Funciona:

- Ela **une (JOIN)** as tabelas **Clientes** (para detalhes do cliente) com **Pedidos** (para o histórico de pedidos).
- A cláusula **WHERE** filtra os pedidos realizados **nos últimos seis meses**.
- Ela **agrupa (GROUP BY)** os resultados por **cliente_id**, **nome** e **email** para garantir que cada cliente apareça apenas uma vez.
- **MAX(p.data_pedido)** é usado para recuperar a **data da compra mais recente** para cada cliente ativo. O **DISTINCT** não é estritamente necessário devido ao **GROUP BY**, mas também não prejudica.

4.3. VW_ESTOQUE_BAIXO

```
CREATE OR REPLACE VIEW vw_estoque_baixo AS
SELECT
    l.id AS livro_id, l.titulo, l.quantidade_estoque, c.nome AS categoria
FROM Livros l
JOIN Categorias c ON l.categoria_id = c.id
WHERE l.quantidade_estoque < 10;
```

Propósito: Está view fornece uma visão rápida dos **livros cujo estoque está abaixo de um limite crítico** (neste caso, menos de 10 unidades). É extremamente

importante para a gestão de inventário, permitindo que você reponha proativamente títulos populares antes que se esgotem e afetem as vendas.

Como Funciona:

- Ela **une (JOIN)** as tabelas **Livros** (para detalhes do livro e quantidade em estoque) com **Categorias** (para incluir o nome da categoria).
- A cláusula **WHERE** é direta: ela seleciona apenas os livros onde a **quantidade_estoque** é **menor que 10**.

5. STORED PROCEDURES

5.1. SP_INSERTIR_LIVRO

```
CREATE PROCEDURE sp_inserir_livro (
    IN p_titulo VARCHAR(255), IN p_autor_id INT, IN p_isbn CHAR(20),
    IN p_editora_id INT, IN p_preco DECIMAL(10,2), IN p_quantidade_estoque INT, IN
p_categoria_id INT
)
BEGIN
    INSERT INTO Livros (titulo, autor_id, isbn, editora_id, preco, quantidade_estoque,
categoria_id)
        VALUES (p_titulo, p_autor_id, p_isbn, p_editora_id, p_preco,
p_quantidade_estoque, p_categoria_id);
END$$
```

Propósito: Esta procedure serve para inserir um novo livro na tabela Livros. Ela encapsula a operação de inserção, tornando-a padronizada e mais fácil de ser chamada pela aplicação ou por outras ferramentas.

- **Parâmetros de Entrada (IN):** Recebe todos os dados necessários para um novo registro na tabela Livros (título, ID do autor, ISBN, ID da editora, preço, quantidade em estoque e ID da categoria).

- **Operação:** Executa um simples comando INSERT na tabela Livros com os valores fornecidos nos parâmetros.

5.2. SP_ATUALIZAR_ESTOQUE

```
CREATE PROCEDURE sp_atualizar_estoque(  
    IN p_livro_id INT, IN p_quantidade_vendida INT  
)  
BEGIN  
    DECLARE v_estoque_atual INT;  
    SELECT quantidade_estoque INTO v_estoque_atual FROM Livros WHERE id =  
p_livro_id;  
  
    IF v_estoque_atual >= p_quantidade_vendida THEN  
        UPDATE Livros SET quantidade_estoque = quantidade_estoque -  
p_quantidade_vendida WHERE id = p_livro_id;  
    ELSE  
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Erro: Estoque  
insuficiente para realizar a venda.';  
    END IF;  
END$$
```

Propósito: Esta procedure tem como função principal atualizar a quantidade em estoque de um livro após uma venda, com um importante tratamento de erro para evitar vendas com estoque insuficiente. É uma excelente prática de segurança e integridade de dados.

Parâmetros: ID do livro e quantidade vendida.

Funcionamento: Verifica se há estoque suficiente. Se sim, subtrai a quantidade vendida; caso contrário, gera um erro informando que o estoque é insuficiente.

5.3. SP_GERAR_RELATORIO_VENDAS (Relatório de Vendas por Categoria)

```
CREATE PROCEDURE sp_gerar_relatorio_vendas(
  IN p_data_inicio DATE, IN p_data_fim DATE
)
BEGIN
  SELECT
    c.nome AS categoria,
    SUM(ip.quantidade) AS total_livros_vendidos,
    SUM(ip.quantidade * ip.preco_unitario) AS receita_total
  FROM Itens_Pedido ip
  INNER JOIN Pedidos p ON ip.pedido_id = p.id
  INNER JOIN Livros l ON ip.livro_id = l.id
  INNER JOIN Categorias c ON l.categoria_id = c.id
  WHERE DATE(p.data_pedido) BETWEEN p_data_inicio AND p_data_fim
  GROUP BY c.nome
  ORDER BY receita_total DESC;
END$$
```

Propósito: Esta procedure gera um relatório de vendas totalizadas **por categoria de livro** dentro de um período de datas específico. É ideal para análises de desempenho de vendas por tipo de livro.

Parâmetros: p_data_inicio (Data de início) e p_data_fim (Data de fim).

Funcionamento: Junta os dados de pedidos, itens, livros e categorias para somar a quantidade e receita por categoria no período, ordenando os resultados pela receita total (do maior para o menor).

5.4. SP_RELATORIO_VENDAS_POR_AUTOR (Relatório de Vendas por Autor)

```
CREATE PROCEDURE sp_relatorio_vendas_por_autor(
  IN data_inicio DATE, IN data_fim DATE
)
BEGIN
  SELECT
    a.nome AS autor,
    SUM(ip.quantidade) AS total_vendido,
    SUM(ip.quantidade * ip.preco_unitario) AS total_receita
  FROM Itens_Pedido ip
  JOIN Livros l ON ip.livro_id = l.id
  JOIN Autores a ON l.autor_id = a.id
```

```

JOIN Pedidos p ON ip.pedido_id = p.id
WHERE DATE(p.data_pedido) BETWEEN data_inicio AND data_fim
GROUP BY a.nome
ORDER BY total_vendido DESC;
END$$

```

Propósito: Esta procedure gera um relatório de vendas totalizadas **por autor** em um período de tempo definido. Ajuda a identificar os autores mais vendidos na plataforma.

Parâmetros: data_inicio (Data de início) e data_fim (Data de fim).

Funcionamento: Similar ao relatório por categoria, mas agrupa e soma a quantidade e receita de vendas por autor no período, ordenando pelo total de livros vendidos (do maior para o menor).

5.5. SP_RELATORIO_VENDAS_POR_PERIODO (Relatório de Vendas por Período Geral)

```

CREATE PROCEDURE sp_relatorio_vendas_por_periodo(
    IN data_inicio DATE,
    IN data_fim DATE
)
BEGIN
    SELECT
        COUNT(DISTINCT p.id) AS total_pedidos, -- Número total de pedidos únicos no
período
        SUM(ip.quantidade) AS total_livros_vendidos, -- Quantidade total de livros
vendidos
        SUM(ip.quantidade * ip.preco_unitario) AS receita_total -- Receita total gerada
    FROM
        Pedidos p
    JOIN
        Itens_Pedido ip ON p.id = ip.pedido_id
    WHERE
        DATE(p.data_pedido) BETWEEN data_inicio AND data_fim;
END$$

```

Propósito: Esta procedure gera um resumo consolidado de vendas dentro de um intervalo de datas, retornando o total de pedidos, a quantidade total de livros vendidos e a receita bruta gerada. É útil para análises de desempenho comercial em períodos específicos.

Parâmetros: data_inicio (Data de início) e data_fim (Data de fim).

Funcionamento: Filtra os registros da tabela Pedidos com base no campo data_pedido, considerando o intervalo entre data_inicio e data_fim. Calculando o total_pedidos: número total de pedidos únicos realizados no período. total_livros_vendidos: soma das quantidades de livros vendidos. receita_total: soma do valor total gerado por todas as vendas no período (quantidade * preço_unitario).

5.6. SP_CADASTRAR_CLIENTE

```
CREATE PROCEDURE sp_cadastrar_cliente(
  IN p_nome VARCHAR(255), IN p_email VARCHAR(255),
  IN p_hash_senha VARCHAR(255), IN p_salt_senha VARCHAR(255), IN p_endereco
  VARCHAR(255)
)
BEGIN
  INSERT INTO Clientes (nome, email, hash_senha, salt_senha, endereco)
  VALUES (p_nome, p_email, p_hash_senha, p_salt_senha, p_endereco);
END$$
```

Propósito: Cadastrar um novo cliente na tabela Clientes.

Parâmetros: Nome, e-mail, hash da senha, salt da senha e endereço do cliente.

Funcionamento: Realizar um INSERT direto na tabela Clientes. É importante destacar que ele espera o hash e o salt da senha, não a senha em texto simples, por segurança.

5.7. SP_CRIAR_PEDIDO (Criação de Pedido com Transação)

```

CREATE PROCEDURE sp_criar_pedido (
  IN p_cliente_id INT UNSIGNED,
  OUT p_pedido_id INT UNSIGNED,
  OUT p_mensagem_status VARCHAR(255)
)
BEGIN
  DECLARE v_num_itens_carrinho INT;
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    ROLLBACK;
    SET p_pedido_id = NULL;
    SET p_mensagem_status = 'Erro ao criar o pedido. Transação desfeita.';
  END;

  START TRANSACTION;

  SELECT COUNT(*) INTO v_num_itens_carrinho
  FROM Carrinho
  WHERE cliente_id = p_cliente_id;

  IF v_num_itens_carrinho = 0 THEN
    SET p_pedido_id = NULL;
    SET p_mensagem_status = 'Carrinho de compras vazio. Não é possível criar um
pedido.';
    ROLLBACK;
  ELSE
    INSERT INTO Pedidos (cliente_id, status)
    VALUES (p_cliente_id, 'Aberto');

    SET p_pedido_id = LAST_INSERT_ID();

    INSERT INTO Itens_Pedido (pedido_id, livro_id, quantidade, preco_unitario)
    SELECT
      p_pedido_id,
      carr.livro_id,
      carr.quantidade,
      liv.preco
    FROM Carrinho carr
    JOIN Livros liv ON carr.livro_id = liv.id
    WHERE carr.cliente_id = p_cliente_id;

    DELETE FROM Carrinho
    WHERE cliente_id = p_cliente_id;

    COMMIT;
    SET p_mensagem_status = 'Pedido criado com sucesso!';
  END IF;
END;

```


Propósito: Esta procedure realiza a criação de um novo pedido para um cliente, de forma transacional e com tratamento de erros. Ela verifica se há itens no carrinho, insere o pedido, transfere os itens do carrinho para a tabela `Itens_Pedido`, limpa o carrinho e confirma a operação.

Parâmetros:

- `p_cliente_id` (ID do cliente que está realizando a compra)
- `p_pedido_id` (retorna o ID do pedido criado)
- `p_mensagem_status` (mensagem informando sucesso ou motivo de falha)

Funcionamento:

1. Verifica se o carrinho do cliente possui itens.
2. Se estiver vazio, cancela a operação e retorna mensagem de erro.
3. Caso contrário:
 - Cria o pedido com status inicial Aberto.
 - Move os itens do carrinho para a tabela `Itens_Pedido`, mantendo o preço do momento da compra.
 - Limpa o carrinho do cliente.
 - Finaliza a transação com COMMIT.
4. Se houver qualquer erro, a transação é desfeita automaticamente via ROLLBACK.

Observação: Esta procedure garante atomicidade e integridade das operações durante o processo de compra, sendo essencial para evitar inconsistências nos dados.

6. STORED FUNCTION

6.1. FN_CALCULAR_DESCONTO

```
CREATE FUNCTION fn_calcular_desconto(p_valor_total DECIMAL(10,2))  
RETURNS DECIMAL(10,2)  
DETERMINISTIC  
BEGIN
```

```

DECLARE v_desconto_calculado DECIMAL(10,2);

IF p_valor_total > 500 THEN
    SET v_desconto_calculado = p_valor_total * 0.10; -- 10% de desconto
ELSEIF p_valor_total > 200 THEN
    SET v_desconto_calculado = p_valor_total * 0.05; -- 5% de desconto
ELSE
    SET v_desconto_calculado = 0.00;
END IF;

RETURN v_desconto_calculado;
END$$

```

Propósito: A função `fn_calcular_desconto` tem como objetivo calcular um valor de desconto a ser aplicado a um determinado `p_valor_total`, seguindo uma lógica de faixas de preço. Isso é útil para implementar políticas de desconto automáticas em um sistema de e-commerce.

Parâmetros:

- `p_valor_total DECIMAL(10,2)`: Este é o parâmetro de entrada da função, representa o valor total de uma compra ou transação para a qual o desconto será calculado. É definido como `DECIMAL(10,2)` para garantir precisão com valores monetários (até 10 dígitos no total, com 2 casas decimais).

Valor de Retorno:

`RETURNS DECIMAL(10,2)`: A função retorna um valor do tipo `DECIMAL(10,2)`, que será o montante do desconto calculado.

7. TRIGGER

```

CREATE DEFINER=CURRENT_USER TRIGGER trg_atualiza_estoque_pedido
AFTER UPDATE ON Pedidos
FOR EACH ROW
BEGIN

```

```

IF NEW.status = 'Enviado' AND OLD.status != 'Enviado' THEN

    UPDATE Livros l

    JOIN Itens_Pedido ip ON l.id = ip.livro_id

    SET l.quantidade_estoque = l.quantidade_estoque - ip.quantidade

    WHERE ip.pedido_id = NEW.id;

ELSEIF OLD.status = 'Enviado' AND NEW.status != 'Enviado' THEN

    UPDATE Livros l

    JOIN Itens_Pedido ip ON l.id = ip.livro_id

    SET l.quantidade_estoque = l.quantidade_estoque + ip.quantidade

    WHERE ip.pedido_id = NEW.id;

END IF;

END$$

```

Propósito: A trigger `trg_atualiza_estoque_pedido` foi criada para garantir a **atualização automática do estoque de livros** sempre que o status de um pedido for alterado. Ela assegura que o estoque reflita corretamente as movimentações de saída (envio) e retorno (cancelamento) dos pedidos, mantendo a integridade dos dados e eliminando a necessidade de atualizações manuais.

Tipo de Trigger:

- **AFTER UPDATE**
- Associada à tabela Pedidos, executa-se automaticamente após a alteração de qualquer linha.

Funcionamento:

- **Quando o pedido é marcado como "Enviado" e antes não estava nesse status:**
O sistema interpreta como uma saída de produtos. A trigger subtrai as quantidades dos livros correspondentes no estoque.

- **Quando o pedido deixa de estar como "Enviado" (ex: passa para "Cancelado"):**

A trigger entende que os livros não foram entregues. Portanto, as quantidades são devolvidas ao estoque.

8. ÍNDICES E OTIMIZAÇÃO

CREATE INDEX idx_livros_categoria_id ON Livros (categoria_id);

Justificativa: Esta coluna é uma chave estrangeira e provavelmente será usada com frequência em consultas para filtrar livros por categoria (ex: "mostrar todos os livros de ficção"). Um índice acelera essas buscas e as operações de **JOIN** com a tabela **Categorias**.

CREATE INDEX idx_livros_autor_id ON Livros (autor_id);

Justificativa: Similar à **categoria_id**, **autor_id** é uma chave estrangeira e será frequentemente usada para encontrar livros de um autor específico ou para realizar **JOINS** com a tabela **Autores**. O índice melhora significativamente o desempenho dessas consultas.

CREATE INDEX idx_livros_preco ON Livros (preco);

Justificativa: Consultas que envolvem faixas de preço (ex: "livros entre R50eR100") ou ordenação por preço (**ORDER BY preco**) se beneficiarão enormemente deste índice, pois o banco de dados não precisará escanear a tabela inteira para encontrar os livros que se encaixam nos critérios de preço.

CREATE INDEX idx_pedidos_cliente_id ON Pedidos (cliente_id);

Justificativa: Esta coluna é uma chave estrangeira e é fundamental para encontrar todos os pedidos feitos por um cliente específico. O índice acelera as consultas que buscam o histórico de compras de um cliente ou que realizam **JOINS** com a tabela **Clientes**.

CREATE INDEX idx_pedidos_data_pedido ON Pedidos (data_pedido);

Justificativa: Consultas que filtram pedidos por período (ex: "pedidos do último mês", como na **vw_livros_mais_vendidos** e **vw_clientes_ativos**) ou que ordenam pedidos por data (**ORDER BY data_pedido**) serão muito mais rápidas com este índice.

CREATE INDEX idx_itens_pedido_pedido_id ON Itens_Pedido (pedido_id);

Justificativa: Esta é uma chave estrangeira e é crucial para recuperar todos os itens de um pedido específico. O índice otimiza o desempenho de consultas que detalham o conteúdo de um pedido.

CREATE INDEX idx_itens_pedido_livro_id ON Itens_Pedido (livro_id);

Justificativa: Sendo uma chave estrangeira, este índice é vital para encontrar todos os pedidos que contêm um determinado livro, o que é útil para análises de vendas por produto (como na **vw_livros_mais_vendidos**).

CREATE INDEX idx_carrinho_cliente_id ON Carrinho (cliente_id);

Justificativa: Permite que o sistema recupere rapidamente todos os itens no carrinho de compras de um cliente específico. Isso é essencial para a funcionalidade do carrinho de compras em um e-commerce.

9. CONTROLE DE ACESSO E SEGURANÇA

Esta seção detalha a criação de usuários no banco de dados `ecommerce_livros_db` e a concessão de permissões específicas para cada um, seguindo o princípio do privilégio mínimo para garantir a segurança.

9.1. CRIAÇÃO DE USUÁRIOS

```
CREATE USER IF NOT EXISTS 'usuario_cliente'@'localhost' IDENTIFIED BY  
'senha_forte_para_cliente_app';  
CREATE USER IF NOT EXISTS 'usuario_tecnico'@'%' IDENTIFIED BY  
'senha_segura_para_tecnico';  
CREATE USER IF NOT EXISTS 'usuario_gerente'@'%' IDENTIFIED BY  
'senha_admin_super_segura';
```

CREATE USER IF NOT EXISTS: Este comando cria um novo usuário no MySQL, mas apenas se ele ainda não existir, evitando erros caso o script seja executado múltiplas vezes.

9.2. DETALHES DOS USUÁRIOS:

9.2.1. usuario_cliente

Usuário: `'usuario_cliente'@'localhost'`

Host: `'localhost'` (indica que este usuário só pode se conectar ao banco de dados a partir do mesmo servidor onde o MySQL está rodando, ideal para uma aplicação web que reside no mesmo servidor que o banco de dados).

Propósito: Este usuário é destinado a ser usado pela aplicação de e-commerce. Suas permissões são as mais restritas, limitando-o às operações necessárias para o funcionamento da loja (ex: navegar no catálogo, gerenciar carrinho, fazer pedidos).

9.2.2. usuario_tecnico

Usuário: 'usuario_tecnico'@'%':

Host: '%' o caractere curinga indica que este usuário pode se conectar de *qualquer* host.

Propósito: Este usuário é para a equipe técnica ou de suporte. Ele tem permissões para gerenciar dados (inserir, atualizar, consultar) no catálogo de livros e status de pedidos, mas não para alterar a estrutura do banco de dados (ex: criar/excluir tabelas).

9.2.3. usuario_gerente

Usuário: 'usuario_gerente'@'%':

Host: '%' pode se conectar de qualquer host

Propósito: Este usuário é o administrador do banco de dados `ecommerce_livros_db`. Ele possui as permissões mais amplas dentro deste banco de dados, incluindo a capacidade de conceder permissões a outros usuários.

10. VALIDAÇÃO DE SEGURANÇA DE SENHAS

10.1. IMPLEMENTAÇÃO DE HASH + SALT

No contexto do projeto, foi adotada a prática de armazenar senhas utilizando **funções de hash com salt**. Essa abordagem é amplamente reconhecida como uma das mais seguras na proteção de credenciais de usuários em sistemas modernos.

A **função de hash** transforma a senha original em uma sequência irreversível de caracteres, enquanto o **salt** é um valor aleatório adicionado à senha antes da aplicação do hash, com o objetivo de impedir ataques baseados em dicionário ou tabelas pré-computadas (como rainbow tables).

10.2. JUSTIFICATIVA E IMPORTÂNCIA NO CONTEXTO REAL

Embora o projeto esteja centrado em um banco de dados relacional e não inclua uma camada de aplicação completa, a estratégia de segurança adotada reconhece um princípio fundamental: **operações sensíveis como hashing de senhas não devem ser responsabilidade do SGBD**.

Isso ocorre por três razões principais:

1. **Responsabilidade da aplicação, não do banco:** O banco de dados é um mecanismo de persistência de dados, e não um ambiente voltado para execução de lógica criptográfica complexa. A geração de hashes e salts deve ocorrer **na camada de aplicação**, que então envia os valores já processados para o banco.
2. **Ausência de funções nativas robustas no SGBD:** A maioria dos SGBDs relacionais, como o MySQL e o PostgreSQL, não possui suporte nativo para algoritmos seguros como **PBKDF2**, **bcrypt** ou **Argon2**, que são projetados especificamente para o armazenamento seguro de senhas.
3. **Boas práticas de arquitetura:** Armazenar somente os resultados (hash + salt) no banco e centralizar a lógica de verificação na aplicação garante maior controle, auditabilidade e flexibilidade de atualização dos métodos de

segurança sem dependência do banco.

Portanto, a presença dos campos `hashSenha` e `salt` na estrutura da tabela `Clientes` ou `Usuarios` não implica que o banco gere esses valores. Eles são inseridos diretamente pela aplicação externa, que implementa o algoritmo de hash seguro e a geração de `salt` aleatório. Essa separação de responsabilidades **reflete o comportamento esperado em sistemas reais**, respeitando os princípios de segurança, modularidade e escalabilidade.

11. CONEXÃO EM AMBIENTE COLABORATIVO (HOST/CLIENTE)

A conexão `host/cliente` em banco de dados refere-se à comunicação entre uma máquina que atua como **servidor** (`host`) e outras que acessam esse servidor como **clientes**. Essa abordagem é comum em ambientes de desenvolvimento colaborativo, onde vários usuários precisam interagir com um banco centralizado.

No contexto do projeto, essa conexão permite que diferentes membros da equipe acessem o mesmo banco de dados a partir de suas máquinas, utilizando ferramentas como o MySQL Workbench. O servidor mantém os dados e executa as requisições, enquanto os clientes apenas consomem os serviços expostos.

Essa arquitetura reproduz o funcionamento típico de sistemas em produção, onde o banco de dados está hospedado em um servidor (físico ou em nuvem), e a aplicação ou usuários remotos se conectam a ele via rede, utilizando IP e credenciais válidas.

A configuração desse ambiente envolve apenas ajustes pontuais no servidor para aceitar conexões externas, mantendo a segurança e o controle de acesso. No projeto, o objetivo principal foi **viabilizar o trabalho em equipe**, permitindo que o banco fosse acessado por diferentes desenvolvedores de forma centralizada, segura e organizada.

12. POLÍTICA DE BACKUP

1. Objetivo

Garantir a continuidade do negócio através de um plano de backup e recuperação robusto, minimizando a perda de dados (RPO - Recovery Point Objective) e o tempo de inatividade do sistema (RTO - Recovery Time Objective).

2. Tipos e Frequência de Backup

Para um e-commerce, a perda de transações é crítica. A estratégia adotada neste projeto utiliza um script (`.bat`) automatizado pelo Agendador de Tarefas do Windows para executar uma rotina mista de backup.

- **Backup Completo (Full Dump):**
 - **O que é:** Uma cópia completa de todo o banco de dados, incluindo tabelas, dados, procedures e triggers.
 - **Ferramenta:** `mysqldump`.
 - **Frequência:** Executado **diariamente**, durante um período de baixa atividade (ex: 02:00 da madrugada). O script automaticamente categoriza e salva o arquivo de acordo com o dia (Diário, Semanal ou Mensal).
- **Backup Incremental (Binary Logs):**
 - **O que é:** O MySQL registra todas as transações (inserts, updates, deletes) em arquivos chamados "Logs Binários". Fazer o backup desses logs permite restaurar o banco a um ponto específico no tempo (Point-in-Time Recovery).
 - **Frequência:** No modelo implementado, os logs binários são rotacionados (`flush-logs`) e copiados para o local de backup **uma vez ao dia**, junto com a execução do backup completo. Isso garante que todas as transações do dia anterior fiquem salvas.

3. Política de Retenção

A política de retenção é gerenciada automaticamente pelo script, que organiza os backups em pastas específicas e aplica regras de limpeza distintas para cada uma.

- **Backups Completos Diários:** Salvos na pasta `\Diario\` e retidos por **30 dias**.
- **Backups Completos de Fim de Semana (Sábado):** Salvos na pasta `\Semana1\` e retidos por **6 meses**.
- **Backups Completos de Fim de Mês:** Salvos na pasta `\Mensal\` e retidos por **2 anos** (para fins de auditoria).
- **Logs Binários (Incrementais):** Salvos na pasta `\LogsBinarios\` e retidos por **7 dias**.

4. Armazenamento e Segurança

Para este projeto, foi adotada uma abordagem de armazenamento local, centralizada e de fácil gerenciamento.

- **Local de Armazenamento:** Todos os backups (completos e incrementais) são salvos em um diretório base no disco local, especificado como `C:\MySQL_Backups`.
- **Organização:** Dentro deste diretório, o script cria automaticamente as subpastas `Diario`, `Semana1`, `Mensal` e `LogsBinarios` para manter os arquivos organizados conforme a política de retenção.
- **Observação:** Para um ambiente de produção real, o próximo passo seria expandir esta política para incluir a "Regra 3-2-1", replicando estes backups para uma mídia diferente e um local externo (off-site), como um serviço de nuvem.

5. Testes de Recuperação (Restore)

Um backup que não foi testado não é confiável. A validação da estratégia de recuperação é uma prática recomendada.

- **Frequência Recomendada:** Recomenda-se que a equipe realize um teste completo de restauração, no mínimo, **trimestralmente**.
- **Procedimento Sugerido:** O backup mais recente deve ser restaurado em um ambiente de homologação (um banco de dados separado, como `ecommerce_restaurado`) para validar a integridade dos dados e confirmar que o procedimento de recuperação funciona conforme o esperado.

13. CONSULTAS DEMONSTRATIVAS

```
SELECT l.titulo, SUM(ip.quantidade) AS total_vendido  
  
FROM Itens_Pedido ip  
  
JOIN Livros l ON ip.livro_id = l.id  
  
GROUP BY l.titulo  
  
ORDER BY total_vendido DESC;
```

```
SELECT c.nome, COUNT(p.id) AS total_pedidos  
  
FROM Pedidos p  
  
JOIN Clientes c ON p.cliente_id = c.id  
  
GROUP BY c.nome  
  
ORDER BY total_pedidos DESC;
```

```
SELECT status, COUNT(*) AS total  
  
FROM Pedidos
```

```
GROUP BY status;
```

14. CONCLUSÃO

A realização deste projeto proporcionou uma experiência prática completa e integradora na aplicação dos conhecimentos adquiridos na disciplina de Tópicos Avançados em Banco de Dados. Ao simular o funcionamento de um sistema real de e-commerce, foi possível consolidar conceitos fundamentais de modelagem relacional, normalização de dados, uso avançado de SQL (DDL, DML, views, procedures, functions, índices), além de práticas de segurança e manutenção de banco de dados.

Destaca-se o desenvolvimento de mecanismos essenciais para qualquer sistema em ambiente produtivo, como controle de acesso por usuários com diferentes níveis de permissão, criptografia de senhas com hash e salt, e um robusto sistema de backup automatizado com política de retenção e possibilidade de restauração pontual via logs binários.

Além da implementação técnica, o projeto também favoreceu a organização do pensamento lógico, a documentação clara e estruturada, e a simulação de ambientes colaborativos, com destaque para a configuração de conexões remotas seguras.

Em síntese, este trabalho cumpriu com excelência os objetivos propostos, demonstrando a viabilidade de aplicar os recursos avançados do MySQL em um cenário próximo à realidade do mercado. Mais do que um exercício técnico, o projeto

serviu como ponte entre a teoria estudada e as exigências práticas enfrentadas por profissionais de banco de dados no mundo real.