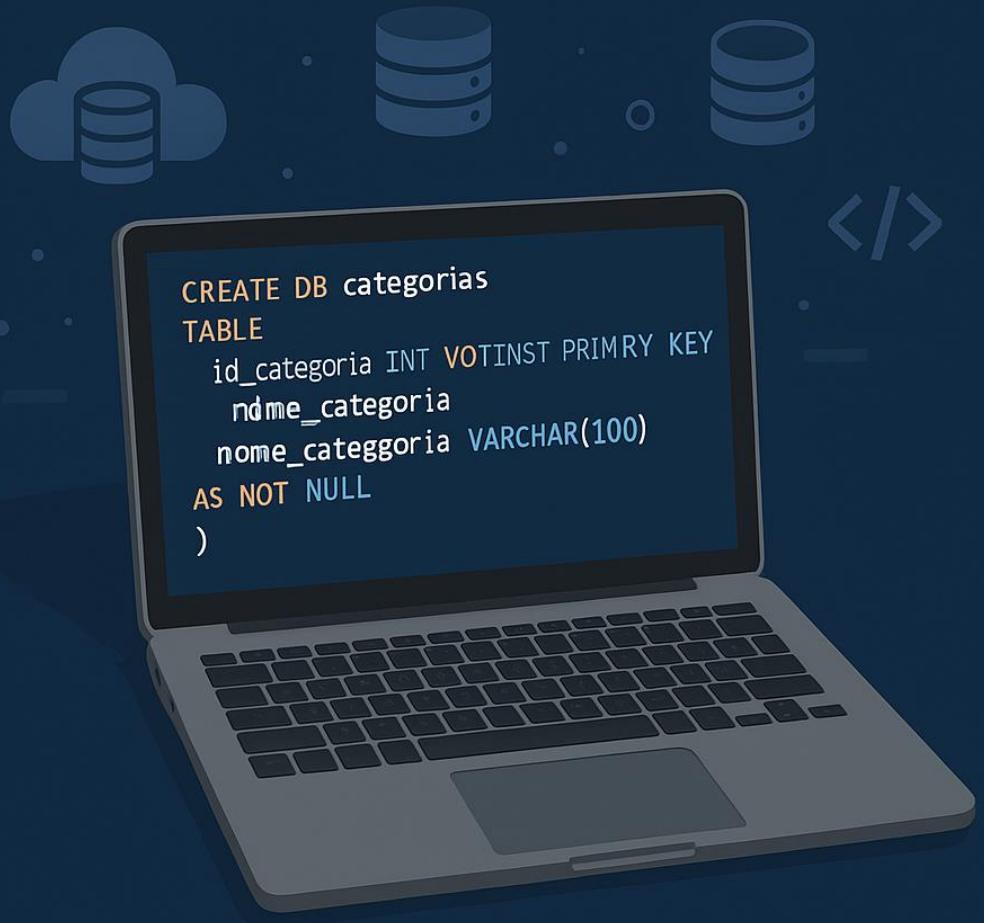


APOSTILA MySQL

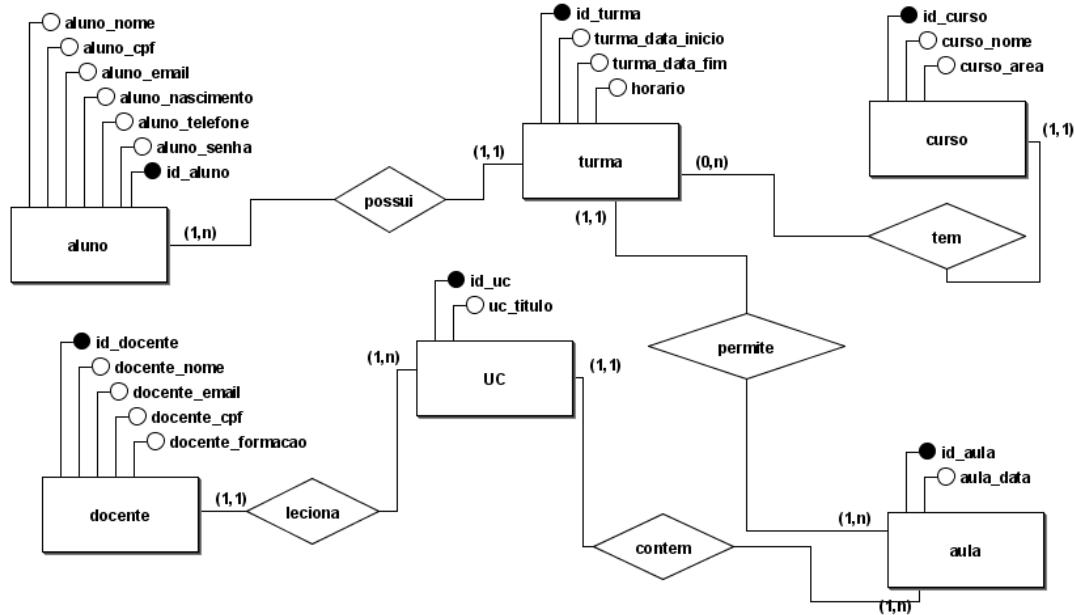


PROFESSOR CELSO
SENAC LAPA TITO

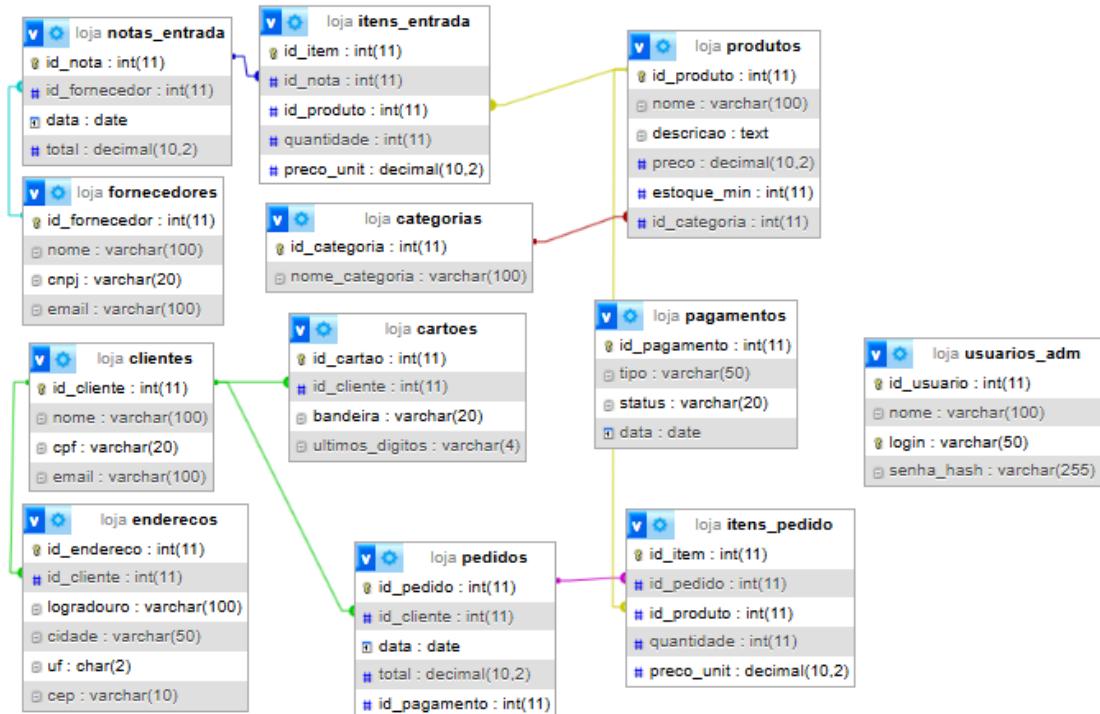
Diagramas Visuais (Modelo Relacional)

Os diagramas abaixo representam os relacionamentos entre as principais tabelas dos projetos estudados nesta apostila.

👉 Diagrama MER - Sistema de Loja



👉 Diagrama DER - Sistema Financeiro



O Que é um SGDB?

SGDB – Sistema Gerenciador de Banco de Dados.

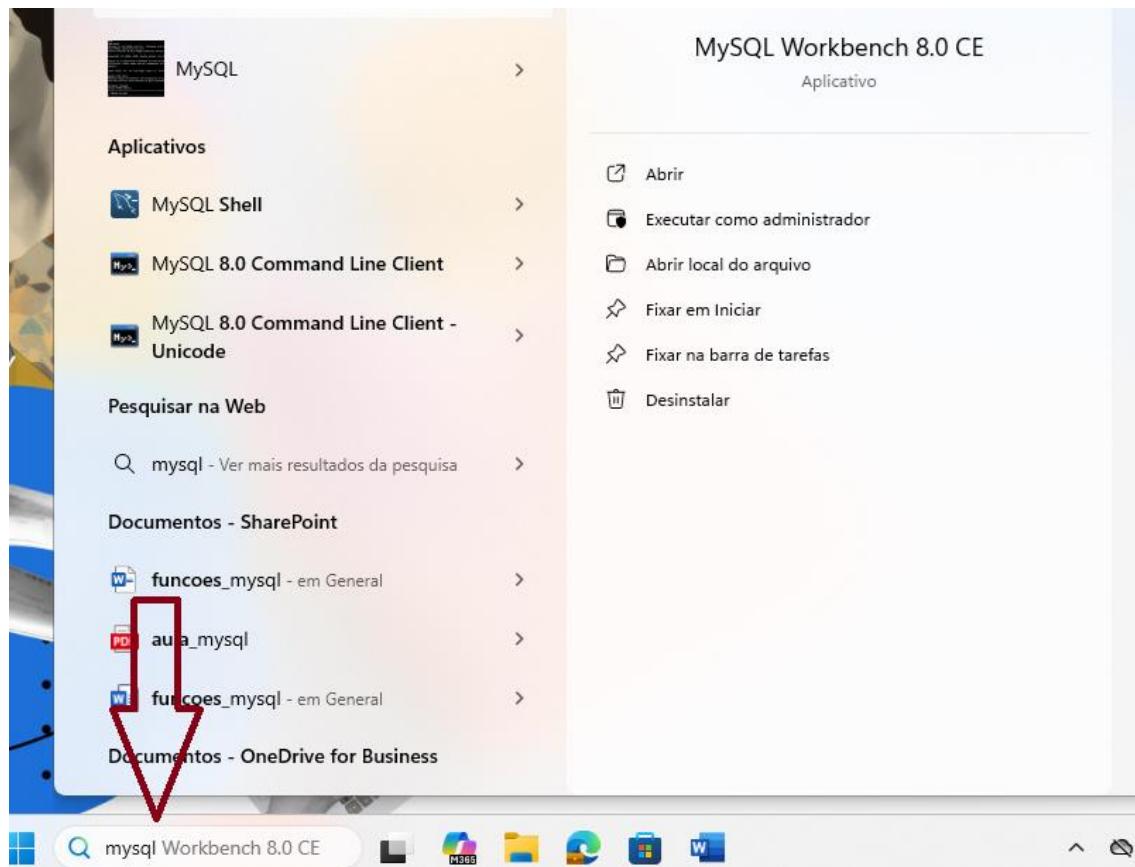
Para se manipular um banco de dados, seja qual for, precisamos de um sistema gerenciador.

Já vimos o gerenciador do Xampp

Hoje vamos conhecer o MySQL Workbench

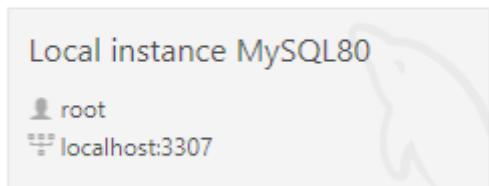
O Gerenciador do Xampp só funciona localmente, já o Workbench acessa uma instância local e também servidores remotos.

Abrir o MySQL Workbench



Selecione a opção abaixo:

MySQL Connections



A senha é senac.

Fundamentos de SQL com MySQL

SQL (Structured Query Language) é a linguagem utilizada para interação com bancos de dados relacionais como o MySQL.

1. Operações CRUD

CRUD é um acrônimo para as quatro operações básicas em um banco de dados:

- CREATE – Inserir dados
- READ – Consultar dados
- UPDATE – Atualizar dados
- DELETE – Remover dados

 Exemplo prático usando a tabela `produtos`:

-- CREATE

```
INSERT INTO produtos (nome_produto, preco, id_categoria)
VALUES ('Mouse Gamer', 99.90, 1);
```

-- READ

```
SELECT * FROM produtos;
```

-- UPDATE

```
UPDATE produtos SET preco = 89.90 WHERE nome_produto =
'Mouse Gamer';
```

-- DELETE

```
DELETE FROM produtos WHERE nome_produto = 'Mouse Gamer';
```

2. Filtros com WHERE, IN, BETWEEN e LIKE

Esses filtros permitem refinar consultas com condições.

-- WHERE com igualdade

```
SELECT * FROM produtos WHERE preco > 100;
```

-- IN

```
SELECT * FROM produtos WHERE id_categoria IN (1, 3);
```

-- BETWEEN

```
SELECT * FROM produtos WHERE preco BETWEEN 50 AND 150;
```

-- LIKE

```
SELECT * FROM produtos WHERE nome_produto LIKE  
'%Camisa%';
```

-- IS NULL

```
SELECT * FROM produtos WHERE id_categoria IS NULL;
```

3. JOINs - Relacionando Tabelas

JOINs permitem unir dados de duas ou mais tabelas relacionadas.

-- INNER JOIN: retorna registros com correspondência nas duas tabelas

```
SELECT p.nome_produto, c.nome_categoria  
FROM produtos p  
INNER JOIN categorias c ON p.id_categoria = c.id_categoria;
```

-- LEFT JOIN: retorna todos os produtos e a categoria, mesmo que nula

```
SELECT p.nome_produto, c.nome_categoria  
FROM produtos p  
LEFT JOIN categorias c ON p.id_categoria = c.id_categoria;
```

-- RIGHT JOIN (pouco usado no MySQL)

-- FULL OUTER JOIN (não nativo no MySQL, mas simulado com UNION)

4. Combinação com UNION

O UNION permite combinar o resultado de duas ou mais consultas SELECT.

```
SELECT nome_produto AS nome FROM produtos
```

UNION

```
SELECT nome_categoria AS nome FROM categorias;
```

5. Agrupamentos e Filtros Avançados

-- GROUP BY com soma

```
SELECT id_categoria, SUM(preco) AS total_categoria
      FROM produtos
     GROUP BY id_categoria;
```

-- HAVING filtra após agrupamento

```
SELECT id_categoria, COUNT(*) AS total
      FROM produtos
     GROUP BY id_categoria
    HAVING total > 2;
```

-- ORDER BY ordena os resultados

```
SELECT * FROM produtos ORDER BY preco DESC;
```



Exercícios Resolvidos - Fundamentos SQL

1. Liste todos os produtos da categoria 'Roupas'
-

```
SELECT p.nome_produto
      FROM produtos p
 JOIN categorias c ON p.id_categoria = c.id_categoria
 WHERE c.nome_categoria = 'Roupas';
```

2. Liste os produtos cujo preço está entre R\$ 50,00 e R\$ 200,00.
-

```
SELECT nome_produto, preco
      FROM produtos
 WHERE preco BETWEEN 50 AND 200;
```

```
CREATE DATABASE IF NOT EXISTS curso_mysql;
```

```
USE curso_mysql;
```

```
-- Tabela de Categorias
```

```
CREATE TABLE categorias (
    id_categoria INT AUTO_INCREMENT PRIMARY KEY,
    nome_categoria VARCHAR(100) NOT NULL
);
```

```
-- Tabela de Produtos
```

```
CREATE TABLE produtos (
    id_produto INT AUTO_INCREMENT PRIMARY KEY,
    nome_produto VARCHAR(100) NOT NULL,
    preco DECIMAL(10,2) NOT NULL,
    id_categoria INT,
    FOREIGN KEY (id_categoria) REFERENCES categorias(id_categoria)
);
```

```
-- Tabela de Vendas
```

```
CREATE TABLE vendas (
    id_venda INT AUTO_INCREMENT PRIMARY KEY,
    data_venda DATE NOT NULL,
    cliente_nome VARCHAR(100) NOT NULL
);
```

-- Tabela de Itens de Vendas

```
CREATE TABLE itens_vendas (
    id_item INT AUTO_INCREMENT PRIMARY KEY,
    id_venda INT,
    id_produto INT,
    quantidade INT NOT NULL,
    preco_unitario DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (id_venda) REFERENCES vendas(id_venda),
    FOREIGN KEY (id_produto) REFERENCES produtos(id_produto)
);
```

-- Inserir Categorias

```
INSERT INTO categorias (nome_categoria) VALUES
('Eletrônicos'),
('Livros'),
('Roupas'),
('Alimentos');
```

-- Inserir Produtos

```
INSERT INTO produtos (nome_produto, preco, id_categoria) VALUES
('Notebook', 3500.00, 1),
('Smartphone', 2500.00, 1),
('Java para Iniciantes', 89.90, 2),
('Camisa Polo', 79.90, 3),
('Calça Jeans', 129.90, 3),
('Arroz 5kg', 22.90, 4),
('Feijão 1kg', 7.90, 4);
```

-- Inserir Vendas

```
INSERT INTO vendas (data_venda, cliente_nome) VALUES  
('2025-06-20', 'Carlos Souza'),  
('2025-06-21', 'Ana Maria'),  
('2025-06-22', 'João Pedro');
```

-- Inserir Itens de Vendas

```
INSERT INTO itens_vendas (id_venda, id_produto, quantidade, preco_unitario)  
VALUES  
(1, 1, 1, 3500.00), -- Carlos comprou 1 Notebook  
(1, 3, 2, 89.90), -- Carlos comprou 2 livros  
(2, 4, 3, 79.90), -- Ana comprou 3 camisas  
(2, 6, 1, 22.90), -- Ana comprou 1 arroz  
(3, 2, 1, 2500.00), -- João comprou 1 smartphone  
(3, 5, 2, 129.90); -- João comprou 2 calças
```

1. Selecione todos os produtos cadastrados, mostrando o nome do produto, o preço e a categoria.

```
SELECT p.nome_produto, p.preco, c.nome_categoria  
FROM produtos p  
JOIN categorias c ON p.id_categoria = c.id_categoria;
```

2. Liste todas as vendas com o nome do cliente e a data da venda.

```
SELECT id_venda, cliente_nome, data_venda
```

```
FROM vendas;
```

3. Mostre os itens vendidos (nome do produto, quantidade e preço unitário).

```
SELECT p.nome_produto, i.quantidade, i.preco_unitario  
FROM itens_vendas i  
JOIN produtos p ON i.id_produto = p.id_produto;
```

4. Calcule o total de cada item vendido (quantidade × preço_unitário), mostrando também o nome do produto.

```
SELECT p.nome_produto, i.quantidade, i.preco_unitario, (i.quantidade *  
i.preco_unitario) AS total_item  
FROM itens_vendas i
```

```
JOIN produtos p ON i.id_produto = p.id_produto;
```

5. Mostre o total de cada venda (soma dos itens), com nome do cliente e data da venda.

```
SELECT v.id_venda, v.cliente_nome, v.data_venda,  
SUM(i.quantidade * i.preco_unitario) AS total_venda  
FROM vendas v  
JOIN itens_vendas i ON v.id_venda = i.id_venda  
GROUP BY v.id_venda, v.cliente_nome, v.data_venda;
```

6. Liste o total vendido por produto.

```
SELECT p.nome_produto, SUM(i.quantidade) AS total_quantidade,  
SUM(i.quantidade * i.preco_unitario) AS total_vendido  
FROM itens_vendas i  
JOIN produtos p ON i.id_produto = p.id_produto  
GROUP BY p.nome_produto;
```

7. Mostre o total de vendas por categoria.

```
SELECT c.nome_categoria,
       SUM(i.quantidade * i.preco_unitario) AS total_categoria
  FROM itens_vendas i
  JOIN produtos p ON i.id_produto = p.id_produto
  JOIN categorias c ON p.id_categoria = c.id_categoria
 GROUP BY c.nome_categoria;
```

8. Liste os produtos que foram vendidos mais de 2 vezes no total.

```
SELECT p.nome_produto, SUM(i.quantidade) AS total_vendido
  FROM itens_vendas i
  JOIN produtos p ON i.id_produto = p.id_produto
 GROUP BY p.nome_produto
 HAVING total_vendido > 2;
```

9. Exiba o nome dos clientes e os produtos que cada um comprou.

```
SELECT v.cliente_nome, p.nome_produto
  FROM vendas v
  JOIN itens_vendas i ON v.id_venda = i.id_venda
  JOIN produtos p ON i.id_produto = p.id_produto
 ORDER BY v.cliente_nome;
```

10. Liste os produtos que nunca foram vendidos.

```
SELECT p.nome_produto
  FROM produtos p
 LEFT JOIN itens_vendas i ON p.id_produto = i.id_produto
 WHERE i.id_produto IS NULL;
```

Criar Banco de Dados: financeiro

```
--  
-- Banco de dados: `financeiro`  
  
--  
  
DELIMITER $$  
  
--  
-- Procedimentos  
  
--  
CREATE DEFINER=`root`@`localhost` PROCEDURE `ContaFuncionarios` (OUT  
`quantidade` INT) SELECT COUNT(*) INTO quantidade FROM funcionarios$$  
  
--  
-- Funções  
  
--  
CREATE DEFINER=`root`@`localhost` FUNCTION `calcular_desconto` (`valor`  
DECIMAL(10,2), `percentual` INT) RETURNS DECIMAL(10,2) RETURN valor - (valor  
* percentual / 100)$$  
  
  
CREATE DEFINER=`root`@`localhost` FUNCTION `fn_verSalario` (`a`  
SMALLINT) RETURNS VARCHAR(60) CHARSET utf8mb4 COLLATE  
utf8mb4_general_ci RETURN  
(SELECT CONCAT('O salario de ', nome, ' é ', salario)  
FROM funcionarios  
WHERE id_funcionario = a)$$  
  
  
CREATE DEFINER=`root`@`localhost` FUNCTION `imc` (`peso`  
DECIMAL(10,2), `altura` DECIMAL(10,2)) RETURNS DECIMAL(10,2) RETURN peso  
/ (altura*altura)$$
```

```
DELIMITER ;
```

```
CREATE TABLE `centro_custos` (
  `id_centro_custos` int(11) NOT NULL,
  `descricao_centro_custos` varchar(50) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;
```

```
INSERT INTO `centro_custos` (`id_centro_custos`, `descricao_centro_custos`)
VALUES
(1, 'Lapa Tito'),
(2, 'Pompeia'),
(3, 'Sipião');
```

```
CREATE TABLE `contato` (
  `nome_fornecedor` varchar(80)
, `celular` varchar(15)
, `email` varchar(100)
);
```

Estrutura stand-in para view `contatof`
(Veja abaixo para a visão atual)

```
CREATE TABLE `contatof` (
  `nome_fornecedor` varchar(80)
, `celular` varchar(15)
, `email` varchar(100)
);
```

```
CREATE TABLE `fornecedores` (
```

```

`id_fornecedor` int(11) NOT NULL,
`nome_fornecedor` varchar(80) NOT NULL,
`cpf_cnpj` varchar(25) NOT NULL,
`celular` varchar(15) NOT NULL,
`email` varchar(100) NOT NULL,
`cep` varchar(10) NOT NULL,
`logradouro` varchar(60) NOT NULL,
`numero` varchar(15) NOT NULL,
`complemento` varchar(15) NOT NULL,
`bairro` varchar(50) NOT NULL,
`cidade` varchar(50) NOT NULL,
`estado` varchar(2) NOT NULL,
`contato` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

```

-- Estrutura para tabela `funcionarios`

```

CREATE TABLE `funcionarios` (
`id_funcionario` int(11) NOT NULL,
`nome` varchar(60) NOT NULL,
`salario` float NOT NULL DEFAULT 0
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

```

```

CREATE TABLE `pagamentos` (
`id_pagamento` int(11) NOT NULL,
`id_fornecedor` int(11) NOT NULL DEFAULT 0,
`data_vcto` date NOT NULL,
`valor` float NOT NULL DEFAULT 0,

```

```

`data_pagto` date NOT NULL,
`valor_pago` float NOT NULL DEFAULT 0,
`descricao` varchar(80) NOT NULL,
`id_tipo_pagto` int(11) NOT NULL DEFAULT 0,
`id_forma_pagto` int(11) NOT NULL DEFAULT 0,
`oculto` int(11) NOT NULL DEFAULT 0,
`id_usuario` int(11) NOT NULL DEFAULT 0,
`data_exclusao` date NOT NULL,
`id_centro_custos` int(11) NOT NULL DEFAULT 0
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

```

```

CREATE TABLE `plano_contas` (
`id_conta` int(11) NOT NULL,
`codigo_conta` varchar(20) NOT NULL,
`descricao_conta` varchar(100) NOT NULL,
`tipo` varchar(20) NOT NULL,
`grupo` varchar(50) DEFAULT NULL,
`ordem` int(11) DEFAULT 1
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;

```

```

CREATE TABLE `tipo_pagamentos` ( `id_tipo_pagto` int(11) NOT NULL,
`descricao_tipo` varchar(40) NOT NULL,
`id_conta` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

```

```
DROP TABLE IF EXISTS `contato`;
```

```

CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY
DEFINER VIEW `contato` AS SELECT `fornecedores`.`nome_fornecedor` AS
`nome_fornecedor`, `fornecedores`.`celular` AS `celular`,
`fornecedores`.`email` AS `email` FROM `fornecedores`;

```

```
DROP TABLE IF EXISTS `contatof`;

CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY
DEFINER VIEW `contatof` AS SELECT `fornecedores`.`nome_fornecedor` AS
`nome_fornecedor`, `fornecedores`.`celular` AS `celular`,
`fornecedores`.`email` AS `email` FROM `fornecedores`;

ALTER TABLE `centro_custos`
ADD PRIMARY KEY(`id_centro_custos`);

ALTER TABLE `fornecedores`
ADD PRIMARY KEY(`id_fornecedor`);

ALTER TABLE `funcionarios`
ADD PRIMARY KEY(`id_funcionario`);

ALTER TABLE `pagamentos`
ADD PRIMARY KEY(`id_pagamento`);

ALTER TABLE `plano_contas`
ADD PRIMARY KEY(`id_conta`);

ALTER TABLE `tipo_pagamentos`
ADD PRIMARY KEY(`id_tipo_pagto`),
ADD KEY `id_conta`(`id_conta`);

ALTER TABLE `centro_custos`
```

```
    MODIFY `id_centro_custos` int(11) NOT NULL AUTO_INCREMENT,  
    AUTO_INCREMENT=4;
```

```
ALTER TABLE `fornecedores`  
    MODIFY `id_fornecedor` int(11) NOT NULL AUTO_INCREMENT,  
    AUTO_INCREMENT=112;
```

```
ALTER TABLE `funcionarios`  
    MODIFY `id_funcionario` int(11) NOT NULL AUTO_INCREMENT,  
    AUTO_INCREMENT=4;
```

```
ALTER TABLE `pagamentos`  
    MODIFY `id_pagamento` int(11) NOT NULL AUTO_INCREMENT,  
    AUTO_INCREMENT=1376;
```

```
ALTER TABLE `plano_contas`  
    MODIFY `id_conta` int(11) NOT NULL AUTO_INCREMENT,  
    AUTO_INCREMENT=11;  
  
ALTER TABLE `tipo_pagamentos`  
    MODIFY `id_tipo_pagto` int(11) NOT NULL AUTO_INCREMENT,  
    AUTO_INCREMENT=47;  
  
ALTER TABLE `tipo_pagamentos`  
    ADD CONSTRAINT `tipo_pagamentos_ibfk_1` FOREIGN KEY (`id_conta`)  
    REFERENCES `plano_contas`(`id_conta`);
```

Tópicos: Functions, Procedures, Views e Triggers

Esta apostila cobre os principais recursos de programação no MySQL: funções definidas pelo usuário (functions), procedimentos armazenados (procedures), visões (views) e gatilhos (triggers). Além das definições, são apresentados exemplos e scripts para uso em sala de aula.

1. FUNCTIONS (Funções)

Funções são blocos de código armazenados no banco de dados que retornam um valor.

Exemplo de função:

```
CREATE FUNCTION calcular_desconto(valor DECIMAL(10,2), percentual INT)
RETURNS DECIMAL(10,2) RETURN valor - (valor * percentual / 100);
```

Uso:

```
SELECT calcular_desconto(100, 10); -- Resultado: 90.00
```

2. PROCEDURES (Procedimentos Armazenados)

Procedimentos armazenados são rotinas que podem executar várias operações, mas não retornam valor diretamente.

Exemplo de procedure:

```
DELIMITER //
CREATE PROCEDURE listar_clientes()
BEGIN
    SELECT * FROM clientes;
```

Execução: CALL listar_clientes();

3. VIEWS (Visões)

Views são consultas salvas no banco de dados, permitindo reutilização e simplificação de queries.

Exemplo de view:

```
CREATE VIEW vendas_totais AS
SELECT cliente_id, SUM(valor) AS total
FROM vendas
GROUP BY cliente_id;
```

Consulta: SELECT * FROM vendas_totais;

4. TRIGGERS (Gatilhos)

Triggers são blocos automáticos que executam quando certos eventos ocorrem em uma tabela.

Exemplo de trigger:

```
CREATE TRIGGER log_insercao
AFTER INSERT ON clientes
FOR EACH ROW

    INSERT INTO log_atividades(descricao, data)
    VALUES (CONCAT('Novo cliente:', NEW.nome), NOW());
```

5. Estrutura de Tabelas para Demonstração

```
CREATE TABLE `funcionarios` (
    `id_funcionario` int(11) NOT NULL,
    `nome` varchar(60) NOT NULL,
    `salario` float NOT NULL DEFAULT 0
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```

```
INSERT INTO `funcionarios` (`id_funcionario`, `nome`, `salario`) VALUES
(1, 'José Ferreira da Silva', 3500),
(2, 'João Saldanha', 5000),
(3, 'Maria da Silva', 3500);
```

```
ALTER TABLE `funcionarios`
ADD PRIMARY KEY (`id_funcionario`);

ALTER TABLE `funcionarios`
MODIFY `id_funcionario` int(11) NOT NULL AUTO_INCREMENT,
AUTO_INCREMENT=4;
```