

# 📊 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

O conteúdo gerado por IA pode estar incorreto.*

👉 Diagrama DER - Sistema Financeiro

*Interface gráfica do usuário, Aplicativo

O conteúdo gerado por IA pode estar incorreto.*

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 instancia local e também servidores remotos.

Abrir o MySQL Workbench

Interface gráfica do usuário, Texto, Aplicativo

O conteúdo gerado por IA pode estar incorreto.

Selecione a opção abaixo:

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

O conteúdo gerado por IA pode estar incorreto.

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()  
  
 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;