UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

CAIO MACEDO
FERNANDO BULIGON ANTUNES
JOSÉ SEBEN
KAÍQUE MEDEIROS LIMA

MEMORIAL DESCRITIVO
Laboratório de Banco de Dados

SANTA HELENA
2025/2

CAIO MACEDO
FERNANDO BULIGON ANTUNES
JOSÉ SEBEN
KAÍQUE MEDEIROS LIMA

MEMORIAL DESCRITIVO
Laboratório de Banco de Dados

Descriptive Memorial - Database Laboratory

Trabalho de Conclusão de Disciplina de Graduação apresentado como requisito para conclusão da disciplina de Laboratório de Banco de Dados do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Docente: Dra. Leiliane Pereira de Rezende

SANTA HELENA
2025/2

LISTA DE ALGORITMOS

LISTA DE FIGURAS

LISTAGEM DE CÓDIGOS FONTE

LISTA DE ABREVIATURAS E SIGLAS

Siglas

ACID	Atomicidade, Consistência, Isolamento e Durabilidade

SUMÁRIO

1 BANCO DE DADOS CONCEITUAL/LÓGICO

A descrição do BD KJCF&Cia, trabalhado durante todo o documento, é apresentado na Seção 1.1. A modelagem por meio do MER é dada na Seção 1.2. A composição do restante do documento é descrita na Seção 1.3.

1.1 Descrição do Banco de Dados

O BD "KJCF&Cia" corresponde à modelagem de um processo de banco de dados para uma empresa de materiais de construção, que trabalha desde produtos básicos para obras, ferramentas e utilidades em geral até móveis. Além disso, a empresa oferece serviços adicionais como entrega e montagem para seus clientes. Por se tratar de uma loja situada em uma cidade pequena, as entregas são realizadas apenas dentro da cidade e nas cidades vizinhas.

As principais informações a serem armazenadas são descritas a seguir:

- Clientes: Cadastro dos clientes que fazem compras na loja.
- Cargos: Responsável por armazenar os cargos dos funcionários.
- Funcionários: Cadastro dos funcionários da empresa com seus dados.
- Categorias: Organização utilizada para separar os produtos por categorias.
- Fornecedores: Cadastro dos fornecedores, armazenando suas informações.
- Produtos: Cadastro dos produtos da empresa, com suas informações detalhadas.
- Estoque: Controle do estoque, registrando os produtos disponíveis em loja.
- Compras: Controle das compras realizadas junto aos fornecedores.
- Vendas: Registro das vendas realizadas, vinculando clientes e funcionários.
- Cidade: Cadastro das cidades relacionadas aos endereços dos clientes.
- Bairro: Cadastro dos bairros, vinculados a uma cidade.
- Endereço: Armazena o endereço completo dos clientes (rua, número, complemento, bairro e cidade).
- Agendamento de Montagem: Controle dos agendamentos de montagens a serem realizadas em produtos vendidos.

- Entrega: Controle das entregas vinculadas às vendas, com data e status.
- Pagamento Cliente: Registro dos pagamentos efetuados pelos clientes, incluindo forma e status.
- Produto Venda: Registro dos itens vendidos em cada venda (produto, quantidade e valor).
- Receita Extra: Registro de receitas adicionais recebidas fora do processo de vendas.

Algumas regras são necessárias para que as restrições de integridade do BD sejam mantidas. As principais regras são descritas a abaixo, destacando-se que, ao modelar, algumas são inseridas devido à normalização do BD.

- Regras Sobre os Produtos:
  - Deve possuir uma categoria associada e um nome;
  - Um produto representa um conjunto de produtos com aquele nome e categoria.
- Regras sobre os Agendamentos de montagem:
  - O Status de montagem deve ser atualizado com frequência
  - Montagens em atraso devem ser priorizadas
  - Apenas funcionários de certos cargos podem efetuar a montagem.
- Regras sobre a venda de produtos:
  - Sempre que um produto é vendido, sua quantidade deve ser alterada no estoque.
- Regras sobre as Entregas:
  - Entregas devem ser efetuadas na ordem de data, datas mais próximas primeiro.
  - Entrega só pode ser realizada se a venda for concluída.
- Regras sobre os Categorias:
  - As categorias devem ser generalizadas. Ex: Produtos de Jardinagem
  - Deve-se evitar categorias repetidas ou muito semelhantes.

1.2 Modelo Entidade-Relacionamento

O MER composto por 17 tabelas é apresentado na Figura 1.1.

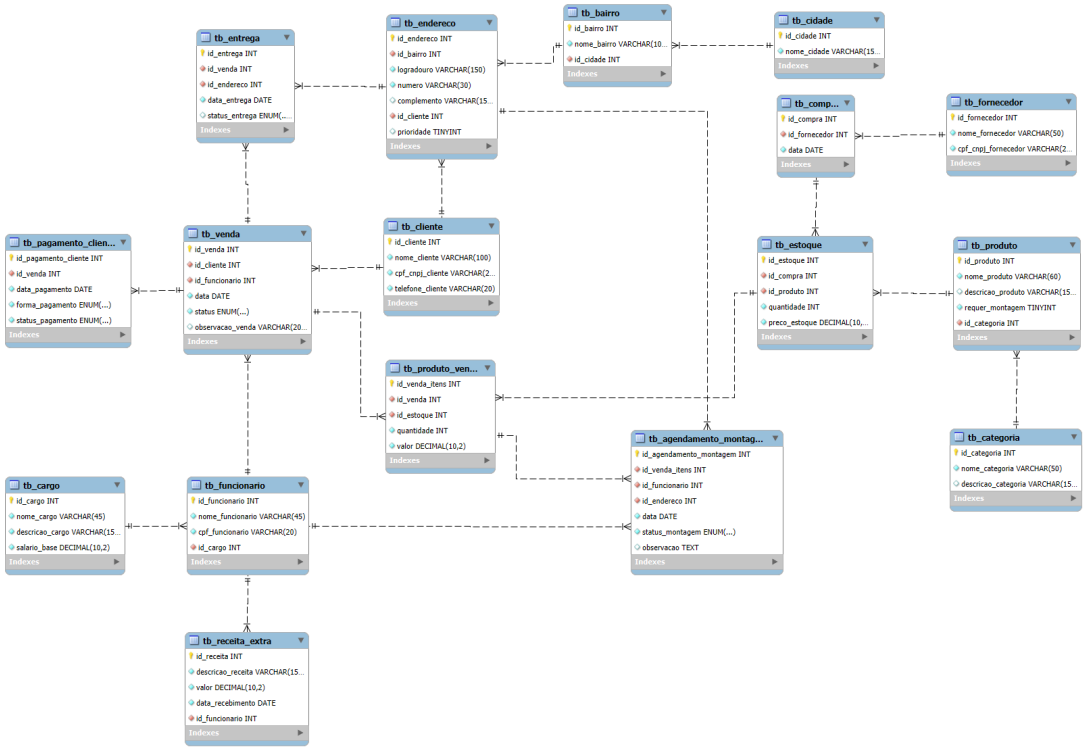

Figura 1.1: Diagrama Modelo-Entidade

1.3 Estrutura do Trabalho

Após a definição conceitual/lógica, o banco deve ser implementado fisicamente em MySQL. Essa implementação, juntamente com a primeira carga de dados, é descrita no Capítulo 2.

Com o banco implementado fisicamente, consultas nos dados podem ser realizadas por meio de `SQL`. O Capítulo apresenta inúmeras consultas, cada uma com uma característica diferente.

Algumas rotinas podem ser definidas para automatizar tarefas e, em alguns casos, melhorar a performance: visões, funções, procedimentos, gatilhos e eventos. Esses recursos são tratados no Capítulo 4.

O acesso aos dados deve ser controlado por segurança. Usuários distintos têm permissões distintas; isso é tratado no Capítulo 5.

Para garantir propriedades ACID, operações de alteração devem ser realizadas em bloco de transação (Capítulo 6).

Por fim, falhas podem ocorrer; tolerâncias e tratamentos são descritos no Capítulo 7.

2 BANCO DE DADOS FÍSICO

A criação do "KJCF&Cia" considerando MySQL é descrita na Seção 2.1 por meio de DDL. A carga inicial é dada na Seção por meio de DML.

2.1 Definição Física

Os scripts responsáveis pela criação física do schema são descritos abaixo. Todas as restrições de integridade (chaves candidatas, primárias e estrangeiras) são consideradas no momento da criação.

Listing 2.1: DDL – Tabela `tb_cliente`

```
1  CREATE TABLE IF NOT EXISTS `mydb`.`tb_cliente` (
2  `id_cliente` INT NOT NULL AUTO_INCREMENT,
3  `nome_cliente` VARCHAR(100) NOT NULL,
4  `cpf_cnpj_cliente` VARCHAR(20) NOT NULL,
5  `telefone_cliente` VARCHAR(20) NOT NULL,
6  PRIMARY KEY (`id_cliente`),
7  UNIQUE INDEX `cpf_cnpj_cliente_UNIQUE` (`cpf_cnpj_cliente` ASC)
       VISIBLE)
8  ENGINE = InnoDB
9  DEFAULT CHARACTER SET = utf8mb4;
```

Listing 2.2: DDL – Tabela `tb_cargo`

```
1  CREATE TABLE IF NOT EXISTS `mydb`.`tb_cargo` (
2  `id_cargo` INT NOT NULL AUTO_INCREMENT,
3  `nome_cargo` VARCHAR(45) NOT NULL,
4  `descricao_cargo` VARCHAR(150) NOT NULL,
5  `salario_base` DECIMAL(10,2) NOT NULL,
6  PRIMARY KEY (`id_cargo`),
7  UNIQUE INDEX `descricao_cargo_UNIQUE` (`descricao_cargo` ASC)
      VISIBLE)
8  ENGINE = InnoDB
9  DEFAULT CHARACTER SET = utf8mb4;
```

Listing 2.3: DDL – Tabela `tb_funcionario`

```
1  CREATE TABLE IF NOT EXISTS `mydb`.`tb_funcionario` (
2  `id_funcionario` INT NOT NULL AUTO_INCREMENT,
3  `nome_funcionario` VARCHAR(45) NOT NULL,
4  `cpf_funcionario` VARCHAR(20) NOT NULL,
```

```
    `id_cargo` INT NOT NULL,
    PRIMARY KEY (`id_funcionario`),
    UNIQUE INDEX `cpf_funcionario_UNIQUE` (`cpf_funcionario` ASC)
       VISIBLE,
    INDEX `id_cargo_idx` (`id_cargo` ASC) VISIBLE,
    CONSTRAINT `fk_funcionario_cargo`
    FOREIGN KEY (`id_cargo`)
    REFERENCES `mydb`.`tb_cargo` (`id_cargo`))
    ENGINE = InnoDB
    DEFAULT CHARACTER SET = utf8mb4;
```

Listing 2.4: DDL – Tabela `tb_venda`

```
    CREATE TABLE IF NOT EXISTS `mydb`.`tb_venda` (
    `id_venda` INT NOT NULL AUTO_INCREMENT,
    `id_cliente` INT NOT NULL,
    `id_funcionario` INT NOT NULL,
    `data` DATE NOT NULL,
    `status` ENUM('pendente', 'concluida', 'cancelada') NOT NULL,
    `observacao_venda` VARCHAR(200) NULL DEFAULT NULL,
    PRIMARY KEY (`id_venda`),
    INDEX `id_cliente_idx` (`id_cliente` ASC) VISIBLE,
    INDEX `id_funcionario_idx` (`id_funcionario` ASC) VISIBLE,
    CONSTRAINT `fk_venda_cliente`
    FOREIGN KEY (`id_cliente`)
    REFERENCES `mydb`.`tb_cliente` (`id_cliente`),
    CONSTRAINT `fk_venda_funcionario`
    FOREIGN KEY (`id_funcionario`)
    REFERENCES `mydb`.`tb_funcionario` (`id_funcionario`))
    ENGINE = InnoDB
    DEFAULT CHARACTER SET = utf8mb4;
```

Listing 2.5: DDL – Tabela `tb_cidade`

```
    CREATE TABLE IF NOT EXISTS `mydb`.`tb_cidade` (
    `id_cidade` INT NOT NULL AUTO_INCREMENT,
    `nome_cidade` VARCHAR(150) NOT NULL,
    PRIMARY KEY (`id_cidade`),
    UNIQUE INDEX `nome_cidade_UNIQUE` (`nome_cidade` ASC) VISIBLE)
    ENGINE = InnoDB
    DEFAULT CHARACTER SET = utf8mb4;
```

Listing 2.6: DDL – Tabela `tb_bairro`

```
CREATE TABLE IF NOT EXISTS `mydb`.`tb_bairro` (
`id_bairro` INT NOT NULL AUTO_INCREMENT,
`nome_bairro` VARCHAR(100) NOT NULL,
`id_cidade` INT NOT NULL,
PRIMARY KEY (`id_bairro`),
INDEX `id_cidade_idx` (`id_cidade` ASC) VISIBLE,
CONSTRAINT `fk_bairro_cidade`
FOREIGN KEY (`id_cidade`)
REFERENCES `mydb`.`tb_cidade` (`id_cidade`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4;
```

Listing 2.7: DDL – Tabela `tb_endereco`

```
CREATE TABLE IF NOT EXISTS `mydb`.`tb_endereco` (
`id_endereco` INT NOT NULL AUTO_INCREMENT,
`id_bairro` INT NOT NULL,
`logradouro` VARCHAR(150) NOT NULL,
`numero` VARCHAR(30) NOT NULL,
`complemento` VARCHAR(150) NULL DEFAULT NULL,
`id_cliente` INT NOT NULL,
`prioridade` TINYINT NULL DEFAULT NULL,
PRIMARY KEY (`id_endereco`),
INDEX `id_cliente_idx` (`id_cliente` ASC) VISIBLE,
INDEX `id_bairro_idx` (`id_bairro` ASC) VISIBLE,
CONSTRAINT `fk_endereco_cliente`
FOREIGN KEY (`id_cliente`)
REFERENCES `mydb`.`tb_cliente` (`id_cliente`),
CONSTRAINT `fk_endereco_bairro`
FOREIGN KEY (`id_bairro`)
REFERENCES `mydb`.`tb_bairro` (`id_bairro`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4;
```

Listing 2.8: DDL – Tabela `tb_agendamento_montagem`

```
CREATE TABLE IF NOT EXISTS `mydb`.`tb_agendamento_montagem` (
`id_agendamento_montagem` INT NOT NULL AUTO_INCREMENT,
`id_venda_itens` INT NOT NULL,
`id_funcionario` INT NOT NULL,
`id_endereco` INT NOT NULL,
`data` DATE NOT NULL,
`status_montagem` ENUM('realizado', 'pendente', 'cancelado')
```

```
    NOT NULL,
    `observacao` TEXT NULL DEFAULT NULL,
    PRIMARY KEY (`id_agendamento_montagem`),
    INDEX `id_venda_itens_idx` (`id_venda_itens` ASC) VISIBLE,
    INDEX `id_funcionario_idx` (`id_funcionario` ASC) VISIBLE,
    INDEX `id_endereco_idx` (`id_endereco` ASC) VISIBLE,
    CONSTRAINT `fk_agendamento_produto_venda`
    FOREIGN KEY (`id_venda_itens`)
    REFERENCES `mydb`.`tb_produto_venda` (`id_venda_itens`),
    CONSTRAINT `fk_agendamento_funcionario`
    FOREIGN KEY (`id_funcionario`)
    REFERENCES `mydb`.`tb_funcionario` (`id_funcionario`),
    CONSTRAINT `fk_agendamento_endereco`
    FOREIGN KEY (`id_endereco`)
    REFERENCES `mydb`.`tb_endereco` (`id_endereco`))
    ENGINE = InnoDB
    DEFAULT CHARACTER SET = utf8mb4;
```

Listing 2.9: DDL – Tabela `tb_categoria`

```
    CREATE TABLE IF NOT EXISTS `mydb`.`tb_categoria` (
    `id_categoria` INT NOT NULL AUTO_INCREMENT,
    `nome_categoria` VARCHAR(50) NOT NULL,
    `descricao_categoria` VARCHAR(150) NULL DEFAULT NULL,
    PRIMARY KEY (`id_categoria`),
    UNIQUE INDEX `nome_categoria_UNIQUE` (`nome_categoria` ASC)
    VISIBLE)
    ENGINE = InnoDB
    DEFAULT CHARACTER SET = utf8mb4;
```

Listing 2.10: DDL – Tabela `tb_fornecedor`

```
    CREATE TABLE IF NOT EXISTS `mydb`.`tb_fornecedor` (
    `id_fornecedor` INT NOT NULL AUTO_INCREMENT,
    `nome_fornecedor` VARCHAR(50) NOT NULL,
    `cpf_cnpj_fornecedor` VARCHAR(20) NOT NULL,
    PRIMARY KEY (`id_fornecedor`),
    UNIQUE INDEX `cpf_cnpj_fornecedor_UNIQUE` (`cpf_cnpj_fornecedor
    ` ASC) VISIBLE)
    ENGINE = InnoDB
    DEFAULT CHARACTER SET = utf8mb4;
```

Listing 2.11: DDL – Tabela `tb_compra`

```
    CREATE TABLE IF NOT EXISTS `mydb`.`tb_compra` (
    `id_compra` INT NOT NULL AUTO_INCREMENT,
    `id_fornecedor` INT NOT NULL,
    `data` DATE NOT NULL,
    PRIMARY KEY (`id_compra`),
    INDEX `id_fornecedor_idx` (`id_fornecedor` ASC) VISIBLE,
    CONSTRAINT `fk_compra_fornecedor`
    FOREIGN KEY (`id_fornecedor`)
    REFERENCES `mydb`.`tb_fornecedor` (`id_fornecedor`))
    ENGINE = InnoDB
    DEFAULT CHARACTER SET = utf8mb4;
```

Listing 2.12: DDL – Tabela `tb_entrega`

```
    CREATE TABLE IF NOT EXISTS `mydb`.`tb_entrega` (
    `id_entrega` INT NOT NULL AUTO_INCREMENT,
    `id_venda` INT NOT NULL,
    `id_endereco` INT NOT NULL,
    `data_entrega` DATE NOT NULL,
    `status_entrega` ENUM('pendente', 'entregue', 'cancelado') NULL
        DEFAULT 'pendente',
    PRIMARY KEY (`id_entrega`),
    INDEX `id_venda_idx` (`id_venda` ASC) VISIBLE,
    INDEX `id_endereco_idx` (`id_endereco` ASC) VISIBLE,
    CONSTRAINT `fk_entrega_venda`
    FOREIGN KEY (`id_venda`)
    REFERENCES `mydb`.`tb_venda` (`id_venda`),
    CONSTRAINT `fk_entrega_endereco`
    FOREIGN KEY (`id_endereco`)
    REFERENCES `mydb`.`tb_endereco` (`id_endereco`))
    ENGINE = InnoDB
    DEFAULT CHARACTER SET = utf8mb4;
```

Listing 2.13: DDL – Tabela `tb_produto`

```
    CREATE TABLE IF NOT EXISTS `mydb`.`tb_produto` (
    `id_produto` INT NOT NULL AUTO_INCREMENT,
    `nome_produto` VARCHAR(60) NOT NULL,
    `descricao_produto` VARCHAR(150) NULL DEFAULT NULL,
    `requer_montagem` TINYINT NOT NULL DEFAULT 0,
    `id_categoria` INT NOT NULL,
    PRIMARY KEY (`id_produto`),
    UNIQUE INDEX `nome_produto_UNIQUE` (`nome_produto` ASC) VISIBLE
```

```
    ,
    INDEX `id_categoria_idx` (`id_categoria` ASC) VISIBLE,
    CONSTRAINT `fk_produto_categoria`
    FOREIGN KEY (`id_categoria`)
    REFERENCES `mydb`.`tb_categoria` (`id_categoria`))
    ENGINE = InnoDB
    DEFAULT CHARACTER SET = utf8mb4;
```

Listing 2.14: DDL – Tabela `tb_estoque`

```
    CREATE TABLE IF NOT EXISTS `mydb`.`tb_estoque` (
    `id_estoque` INT NOT NULL AUTO_INCREMENT,
    `id_compra` INT NOT NULL,
    `id_produto` INT NOT NULL,
    `quantidade` INT NOT NULL,
    `preco_estoque` DECIMAL(10,2) NOT NULL,
    PRIMARY KEY (`id_estoque`),
    INDEX `id_compra_idx` (`id_compra` ASC) VISIBLE,
    INDEX `id_produto_idx` (`id_produto` ASC) VISIBLE,
    CONSTRAINT `fk_estoque_compra`
    FOREIGN KEY (`id_compra`)
    REFERENCES `mydb`.`tb_compra` (`id_compra`),
    CONSTRAINT `fk_estoque_produto`
    FOREIGN KEY (`id_produto`)
    REFERENCES `mydb`.`tb_produto` (`id_produto`))
    ENGINE = InnoDB
    DEFAULT CHARACTER SET = utf8mb4;
```

Listing 2.15: DDL – Tabela `tb_pagamento_cliente`

```
    CREATE TABLE IF NOT EXISTS `mydb`.`tb_pagamento_cliente` (
    `id_pagamento_cliente` INT NOT NULL AUTO_INCREMENT,
    `id_venda` INT NOT NULL,
    `data_pagamento` DATE NOT NULL,
    `forma_pagamento` ENUM('cartao_credito', 'cartao_debito', 'pix'
       , 'boleto', 'dinheiro') NOT NULL,
    `status_pagamento` ENUM('pendente', 'pago', 'cancelado', '
       atrasado') NOT NULL,
    PRIMARY KEY (`id_pagamento_cliente`),
    INDEX `id_venda_idx` (`id_venda` ASC) VISIBLE,
    CONSTRAINT `fk_pagamento_venda`
    FOREIGN KEY (`id_venda`)
    REFERENCES `mydb`.`tb_venda` (`id_venda`))
```

```
12      ENGINE = InnoDB
13      DEFAULT CHARACTER SET = utf8mb4;
```

Listing 2.16: DDL – Tabela `tb_produto_venda`

```
1       CREATE TABLE IF NOT EXISTS `mydb`.`tb_produto_venda` (
2       `id_venda_itens` INT NOT NULL AUTO_INCREMENT,
3       `id_venda` INT NOT NULL,
4       `id_estoque` INT NOT NULL,
5       `quantidade` INT NOT NULL,
6       `valor` DECIMAL(10,2) NOT NULL,
7       PRIMARY KEY (`id_venda_itens`),
8       INDEX `fk_venda_idx` (`id_venda` ASC) VISIBLE,
9       INDEX `fk_estoque_idx` (`id_estoque` ASC) VISIBLE,
10      CONSTRAINT `fk_produto_venda_to_venda`
11      FOREIGN KEY (`id_venda`)
12      REFERENCES `mydb`.`tb_venda` (`id_venda`),
13      CONSTRAINT `fk_produto_venda_to_estoque`
14      FOREIGN KEY (`id_estoque`)
15      REFERENCES `mydb`.`tb_estoque` (`id_estoque`))
16      ENGINE = InnoDB
17      DEFAULT CHARACTER SET = utf8mb4;
```

Listing 2.17: DDL – Tabela `tb_receita_extra`

```
1       CREATE TABLE IF NOT EXISTS `mydb`.`tb_receita_extra` (
2       `id_receita` INT NOT NULL AUTO_INCREMENT,
3       `descricao_receita` VARCHAR(150) NOT NULL,
4       `valor` DECIMAL(10,2) NOT NULL,
5       `data_recebimento` DATE NOT NULL,
6       `id_funcionario` INT NOT NULL,
7       PRIMARY KEY (`id_receita`),
8       INDEX `id_funcionario_idx` (`id_funcionario` ASC) VISIBLE,
9       CONSTRAINT `fk_receita_funcionario`
10      FOREIGN KEY (`id_funcionario`)
11      REFERENCES `mydb`.`tb_funcionario` (`id_funcionario`))
12      ENGINE = InnoDB
13      DEFAULT CHARACTER SET = utf8mb4;
```

2.2 Carga de Dados Iniciais

Após a criação do banco, uma carga inicial é inserida por meio de DML:

Listing 2.18: DML – Tabela `tb_cargo`

```
INSERT INTO tb_cargo (id_cargo, nome_cargo, descricao_cargo,
   salario_base) VALUES
(1,'Vendedor','Responsável pelo atendimento e vendas em loja'
   ,2000.00),
(2,'Montador','Realiza montagem de móveis',2200.00),
(3,'Gerente','Gerencia a equipe e as metas da loja',4500.00);
```

Listing 2.19: DML – Tabela `tb_funcionario`

```
INSERT INTO tb_funcionario (id_funcionario, nome_funcionario,
   cpf_funcionario, id_cargo) VALUES
(1,'José Seben','00000000000',3),
(2,'Caio Macedo','11111111111',1),
(3,'Kaique Lima','22222222222',2),
(4,'Fernando Buligon','33333333333',1);
```

Listing 2.20: DML – Tabela `tb_cliente`

```
INSERT INTO tb_cliente (id_cliente, nome_cliente,
   cpf_cnpj_cliente, telefone_cliente) VALUES
(1,'João Oliveira','00000000000','(00) 00000-0000'),
(2,'Maria Santos','11111111111','(11) 11111-1111'),
(3,'Eduardo Costa','22222222222','(22) 22222-2222'),
(4,'Gabriela Barros','33333333333','(33) 33333-3333'),
(5,'Pedro Cachoeira','44444444444','(44) 44444-4444');
```

Listing 2.21: DML – Tabela `tb_cidade`

```
INSERT INTO tb_cidade (id_cidade, nome_cidade) VALUES
(1,'Santa Helena'),
(2,'Sao Clemente'),
(3,'Medianeira'),
(4,'Subsede'),
(5,'Entre Rios');
```

Listing 2.22: DML – Tabela `tb_bairro`

```
INSERT INTO tb_bairro (id_bairro, nome_bairro, id_cidade)
   VALUES
(1,'Centro',1),
(2,'Centro',2),
(3,'Centro',3),
(4,'Centro',4),
(5,'Centro',5);
```

Listing 2.23: DML – Tabela `tb_endereco`

```
INSERT INTO tb_endereco (id_endereco, id_bairro, logradouro,
   numero, complemento, id_cliente, prioridade) VALUES
(1,1,'Rua Xaxim','123',NULL,1,1),
(2,2,'Av. Brasil','321',NULL,2,1),
(3,3,'Rua Aroeira','231',NULL,3,1),
(4,4,'Rua Argentina','213',NULL,4,1),
(5,4,'Rua Argentina','312',NULL,5,1);
```

Listing 2.24: DML – Tabela `tb_categoria`

```
INSERT INTO tb_categoria (id_categoria, nome_categoria,
   descricao_categoria) VALUES
(1,'Ferramentas','Makita, martelo, pá...'),
(2,'Mobilia','Guarda-roupa, mesa, bancada...'),
(3,'Materiais','Areia, pedra, prego...');
```

Listing 2.25: DML – Tabela `tb_fornecedor`

```
INSERT INTO tb_fornecedor (id_fornecedor, nome_fornecedor,
   cpf_cnpj_fornecedor) VALUES
(1,'Kaique Madeiras','22222222222222'),
(2,'João das Areias','33333333333333'),
(3,'Pedro Pedradas','44444444444444'),
(4,'Rafael Pregos','55555555555555'),
(5,'Raul Rolamentos','66666666666666');
```

Listing 2.26: DML – Tabela `tb_produto`

```
INSERT INTO tb_produto (id_produto, nome_produto,
   descricao_produto, requer_montagem, id_categoria) VALUES
(1,'Makita 5007N','Serra Circular 7-1/4 1800W',0,1),
(2,'Madeira Pinus','Madeira boa',0,3),
(3,'Mesa Retangular','Mesa de madeira',1,2),
(4,'Pedra Brita','Essa é da boa',0,3),
(5,'Areia Média','Funciona bem',0,3);
```

Listing 2.27: DML – Tabela `tb_compra`

```
INSERT INTO tb_compra (id_compra, id_fornecedor, data) VALUES
(1,1,'2025-08-10'),
(2,1,'2025-08-11'),
(3,1,'2025-08-12'),
(4,1,'2025-08-13');
```

Listing 2.28: DML – Tabela `tb_estoque`

```
INSERT INTO tb_estoque (id_estoque, id_compra, id_produto,
   quantidade, preco_estoque) VALUES
(1,1,1, 50, 793.78),
(2,2,2,100, 120.00),
(3,2,3, 20, 450.00),
(4,3,4,200,  35.00),
(5,4,5,180,  30.00);
```

Listing 2.29: DML – Tabela `tb_venda`

```
INSERT INTO tb_venda (id_venda, id_cliente, id_funcionario,
   data, status, observacao_venda) VALUES
(1,1,2,'2025-08-15','concluida','Cara␣saiu␣feliz'),
(2,2,2,'2025-08-16','pendente','Aguardando␣pagamento'),
(3,3,4,'2025-08-17','concluida','Sucesso'),
(4,4,4,'2025-08-18','cancelada','Cartão␣não␣passou');
```

Listing 2.30: DML – Tabela `tb_produto_venda`

```
INSERT INTO tb_produto_venda (id_venda_itens, id_venda,
   id_estoque, quantidade, valor) VALUES
(1,1,1,189,793.78),
(2,2,1,  1,793.78),
(3,3,1,  1,793.78),
(4,4,1,  1,793.78);
```

Listing 2.31: DML – Tabela `tb_pagamento_cliente`

```
INSERT INTO tb_pagamento_cliente (id_pagamento_cliente,
   id_venda, data_pagamento, forma_pagamento, status_pagamento)
    VALUES
(1,1,'2025-08-15','pix','pago'),
(2,2,'2025-08-17','boleto','pendente'),
(3,3,'2025-08-17','boleto','pago'),
(4,4,'2025-08-18','cartao_credito','cancelado');
```

Listing 2.32: DML – Tabela `tb_entrega`

```
INSERT INTO tb_entrega (id_entrega, id_venda, id_endereco,
   data_entrega, status_entrega) VALUES
(1,1,1,'2025-08-16','entregue'),
(2,2,2,'2025-08-20','pendente'),
(3,3,3,'2025-08-18','entregue');
```

Listing 2.33: DML – Tabela `tb_agendamento_montagem`

```
INSERT INTO tb_agendamento_montagem (id_agendamento_montagem,
   id_venda_itens, id_funcionario, id_endereco, data,
   status_montagem, observacao) VALUES
(1,1,2,1,'2025-08-16','realizado','Cliente chato'),
(2,2,2,2,'2025-08-21','pendente','No aguardo da chegada dos
   materiais'),
(3,3,2,3,'2025-08-19','realizado','Sucesso');
```

Listing 2.34: DML – Tabela `tb_receita_extra`

```
INSERT INTO tb_receita_extra (id_receita, descricao_receita,
   valor, data_recebimento, id_funcionario) VALUES
(1,'Rapaz merece',350.00,'2025-08-19',2),
(2,'Bateu o recorde em montagem de balcão',120.00,'2025-08-19'
   ,3),
(3,'Comprou makita',150.00,'2025-08-20',1);
```

3 CONSULTAS DE DADOS NO BD

A linguagem SQL permite diferentes recursos para recuperar informações. As seções abaixo listam os tipos de consultas (exemplos a completar).

3.1 Restrições de Linha

Listing 3.1: SELECT – Tabela`tb_cliente`

```
1        SELECT * FROM tb_cliente
2        WHERE cpf_cnpj_cliente IN ('00000000000','11111111111');
```

Listing 3.2: SELECT – Tabela `tb_cargo`

```
1        SELECT * FROM tb_cargo
2        WHERE salario_base > 3000.00;
```

Listing 3.3: SELECT – Tabela `tb_funcionario`

```
1        SELECT * FROM tb_funcionario
2        WHERE id_cargo = 1;
```

Listing 3.4: SELECT – Tabela `tb_venda`

```
1        SELECT * FROM tb_venda
2        WHERE status = 'concluida'
```

Listing 3.5: SELECT – Tabela `tb_cidade`

```
1        SELECT * FROM tb_cidade
2        WHERE nome_cidade IN ('Santa␣Helena','Medianeira');
```

Listing 3.6: SELECT – Tabela `tb_bairro`

```
1        SELECT * FROM tb_bairro
2        WHERE nome_bairro = 'Centro'
3        AND id_cidade IN (4,5);
```

Listing 3.7: SELECT – Tabela `tb_endereco`

```
1        SELECT * FROM tb_endereco
2        WHERE id_cliente IN (1,5)
3        AND complemento IS NULL;
```

Listing 3.8: SELECT – Tabela `tb_agendamento_montagem`

```
SELECT * FROM tb_agendamento_montagem
WHERE status_montagem = 'realizado'
AND data >= '2025-08-19';
```

Listing 3.9: SELECT – Tabela `tb_categoria`

```
SELECT * FROM tb_categoria
WHERE nome_categoria <> 'Ferramentas';
```

Listing 3.10: SELECT – Tabela `tb_fornecedor`

```
SELECT * FROM tb_fornecedor
WHERE cpf_cnpj_fornecedor LIKE '33%';
```

Listing 3.11: SELECT – Tabela `tb_compra`

```
SELECT * FROM tb_compra
WHERE id_fornecedor = 1
AND data >= '2025-08-12';
```

Listing 3.12: SELECT – Tabela `tb_entrega`

```
SELECT * FROM tb_entrega
WHERE status_entrega = 'pendente'
OR data_entrega > '2025-08-17';
```

Listing 3.13: SELECT – Tabela `tb_produto`

```
SELECT * FROM tb_produto
WHERE id_categoria = 3
AND descricao_produto IS NOT NULL;
```

Listing 3.14: SELECT – Tabela `tb_estoque`

```
SELECT * FROM tb_estoque
WHERE quantidade >= 100
AND preco_estoque <= 120.00;
```

Listing 3.15: SELECT – Tabela `tb_pagamento_cliente`

```
SELECT * FROM tb_pagamento_cliente
WHERE forma_pagamento IN ('pix','boleto')
AND status_pagamento = 'pago';
```

Listing 3.16: SELECT – Tabela `tb_produto_venda`

```
SELECT * FROM tb_produto_venda
WHERE id_produto = 1
AND quantidade >= 100;
```

Listing 3.17: SELECT – Tabela `tb_receita_extra`

```
SELECT * FROM tb_receita_extra
WHERE data_recebimento BETWEEN '2025-08-19' AND '2025-08-20'
AND valor >= 150.00;
```

3.2 Funções de Linha

Listing 3.18: SELECT – String: maiusculização e tamanho do texto

```
SELECT id_categoria,
       nome_categoria,
       UPPER(nome_categoria) AS nome_maiusculo,
       LENGTH(nome_categoria) AS tam
FROM tb_categoria;
```

Listing 3.19: SELECT – String: máscara simples de CPF (11 dígitos)

```
SELECT id_cliente,
       nome_cliente,
       CONCAT(
         SUBSTRING(cpf_cnpj_cliente,1,3),'.',
         SUBSTRING(cpf_cnpj_cliente,4,3),'.',
         SUBSTRING(cpf_cnpj_cliente,7,3),'-',
         SUBSTRING(cpf_cnpj_cliente,10,2)
       ) AS cpf_formatado
FROM tb_cliente
WHERE CHAR_LENGTH(cpf_cnpj_cliente) = 11;
```

Listing 3.20: SELECT – String + NULL-safe: endereço completo (COALESCE)

```
SELECT id_endereco,
       CONCAT(
         logradouro, ', ', numero,
         COALESCE(CONCAT(' - ', complemento), '')
       ) AS endereco_completo
FROM tb_endereco;
```

Listing 3.21: SELECT – Data: formatação BR e diferença de dias

```
SELECT id_venda,
       DATE_FORMAT(data, '%d/%m/%Y') AS data_br,
       DATEDIFF(CURDATE(), data)     AS dias_desde_a_venda
FROM tb_venda;
```

Listing 3.22: SELECT – Numérica + condicional: total por item e classificação

```
SELECT id_venda_itens,
       quantidade,
       valor,
       ROUND(quantidade * valor, 2) AS total_item,
       CASE
         WHEN quantidade * valor >= 1000 THEN 'alto'
         WHEN quantidade * valor >= 100  THEN 'médio'
         ELSE 'baixo'
       END AS faixa_valor
FROM tb_produto_venda;
```

3.3 Junções de Tabelas

Listing 3.23: SELECT – INNER JOIN: produto e sua categoria

```
SELECT p.id_produto,
       p.nome_produto,
       c.nome_categoria
FROM tb_produto p
INNER JOIN tb_categoria c
        ON c.id_categoria = p.id_categoria;
```

Listing 3.24: SELECT – INNER JOIN em cadeia: venda com cliente e total

```
SELECT v.id_venda,
       cl.nome_cliente,
       SUM(pv.quantidade * pv.valor) AS total_venda
FROM tb_venda v
INNER JOIN tb_cliente cl
        ON cl.id_cliente = v.id_cliente
INNER JOIN tb_produto_venda pv
        ON pv.id_venda = v.id_venda
GROUP BY v.id_venda, cl.nome_cliente;
```

Listing 3.25: SELECT – LEFT JOIN: compras com (ou sem) itens de estoque

```
        SELECT co.id_compra,
               f.nome_fornecedor,
               e.id_estoque,
               e.quantidade
        FROM tb_compra co
        INNER JOIN tb_fornecedor f
                ON f.id_fornecedor = co.id_fornecedor
        LEFT JOIN tb_estoque e
               ON e.id_compra = co.id_compra
        ORDER BY co.id_compra;
```

Listing 3.26: SELECT – Anti-join (LEFT + IS NULL): cidades sem bairros

```
        SELECT ci.id_cidade,
               ci.nome_cidade
        FROM tb_cidade ci
        LEFT JOIN tb_bairro b
               ON b.id_cidade = ci.id_cidade
        WHERE b.id_bairro IS NULL;
```

Listing 3.27: SELECT – SELF JOIN: pares de funcionários do mesmo cargo

```
        SELECT f1.nome_funcionario AS funcionario_1,
               f2.nome_funcionario AS funcionario_2,
               c.nome_cargo
        FROM tb_funcionario f1
        INNER JOIN tb_funcionario f2
                ON f1.id_cargo = f2.id_cargo
               AND f1.id_funcionario < f2.id_funcionario
        INNER JOIN tb_cargo c
                ON c.id_cargo = f1.id_cargo;
```

3.4 Operadores de Conjunto

Consultas que combinam conjuntos de linhas podem utilizar `UNION` (remoção de duplicatas) e `UNION ALL` (mantém duplicatas). Em MySQL, operações como `INTERSECT` e `EXCEPT` podem ser obtidas por equivalentes com `JOIN`, `EXISTS` ou `NOT EXISTS`.

Listing 3.28: SELECT – UNION: nomes de clientes e fornecedores sem duplicatas

```
        SELECT nome_cliente AS nome, 'cliente' AS origem
        FROM tb_cliente
        UNION
        SELECT nome_fornecedor AS nome, 'fornecedor' AS origem
        FROM tb_fornecedor;
```

Listing 3.29: SELECT – UNION ALL: contabiliza todas as ocorrências (com duplicatas)

```
        SELECT nome_cliente AS nome
        FROM tb_cliente
        UNION ALL
        SELECT nome_fornecedor AS nome
        FROM tb_fornecedor;
        -- Útil quando se deseja manter contagens exatas de ocorrê
           ncias
```

Listing 3.30: SELECT – INTERSECT (equivalente): nomes presentes em cliente e fornecedor

```
        SELECT c.nome_cliente AS nome
        FROM tb_cliente c
        WHERE EXISTS (
            SELECT 1
            FROM tb_fornecedor f
            WHERE f.nome_fornecedor = c.nome_cliente
        );
        -- Equivalente ao INTERSECT dos nomes (quando houver
           coincidência literal)
```

Listing 3.31: SELECT – EXCEPT (equivalente): clientes que nunca compraram

```
        SELECT c.id_cliente, c.nome_cliente
        FROM tb_cliente c
        WHERE NOT EXISTS (
            SELECT 1
            FROM tb_venda v
            WHERE v.id_cliente = c.id_cliente
        );
        -- Equivalente ao "clientes" EXCEPT "clientes com venda"
```

Listing 3.32: SELECT – UNION: cidades e bairros (rótulo por tipo)

```
        SELECT nome_cidade AS nome, 'cidade' AS tipo
        FROM tb_cidade
        UNION
        SELECT nome_bairro AS nome, 'bairro' AS tipo
        FROM tb_bairro;
```

3.5 Sub-Consultas

Subconsultas podem ser escalares (retornam um único valor), de conjunto (`IN`), correlacionadas (`EXISTS/NOT EXISTS`) ou em `FROM` (tabelas derivadas). Abaixo, exemplos aplicados ao schema definido.

Listing 3.33: SELECT – Subconsulta escalar: maior salário-base de cargo

```
1         SELECT nome_cargo, salario_base
2         FROM tb_cargo
3         WHERE salario_base = (
4             SELECT MAX(salario_base) FROM tb_cargo
5         );
```

Listing 3.34: SELECT – IN: clientes com ao menos uma venda concluída

```
1         SELECT c.id_cliente, c.nome_cliente
2         FROM tb_cliente c
3         WHERE c.id_cliente IN (
4             SELECT v.id_cliente
5             FROM tb_venda v
6             WHERE v.status = 'concluida'
7         );
```

Listing 3.35: SELECT – EXISTS (correlata): fornecedores que já tiveram compras

```
1         SELECT f.id_fornecedor, f.nome_fornecedor
2         FROM tb_fornecedor f
3         WHERE EXISTS (
4             SELECT 1
5             FROM tb_compra co
6             WHERE co.id_fornecedor = f.id_fornecedor
7         );
```

Listing 3.36: SELECT – Subconsulta correlata com agregação: vendas com total > 1000

```
1         SELECT v.id_venda, v.data, v.status
2         FROM tb_venda v
3         WHERE (
4             SELECT SUM(pv.valor * pv.quantidade)
5             FROM tb_produto_venda pv
6             WHERE pv.id_venda = v.id_venda
7         ) > 1000.00;
```

Listing 3.37: SELECT – Tabela derivada (FROM): top categorias por itens vendidos

```
        SELECT d.id_categoria, d.nome_categoria, d.qtd_itens
        FROM (
            SELECT pr.id_categoria,
                   ca.nome_categoria,
                   SUM(pv.quantidade) AS qtd_itens
            FROM tb_produto_venda pv
            JOIN tb_estoque es  ON es.id_estoque  = pv.id_estoque
            JOIN tb_produto pr  ON pr.id_produto  = es.id_produto
            JOIN tb_categoria ca ON ca.id_categoria = pr.
               id_categoria
            GROUP BY pr.id_categoria, ca.nome_categoria
        ) AS d
        ORDER BY d.qtd_itens DESC
        LIMIT 5;
```

Listing 3.38: SELECT – NOT EXISTS (correlata): produtos que requerem montagem e nunca foram vendidos

```
        SELECT p.id_produto, p.nome_produto
        FROM tb_produto p
        WHERE p.requer_montagem = 1
          AND NOT EXISTS (
              SELECT 1
              FROM tb_estoque e
              JOIN tb_produto_venda pv ON pv.id_estoque = e.
                 id_estoque
              WHERE e.id_produto = p.id_produto
          );
```

Listing 3.39: SELECT – Subconsulta escalar em projeção: total (R$) por venda

```
        SELECT v.id_venda,
               v.status,
               (
                 SELECT SUM(pv.valor * pv.quantidade)
                 FROM tb_produto_venda pv
                 WHERE pv.id_venda = v.id_venda
               ) AS total_venda
        FROM tb_venda v;
```

3.6 Agregação e Funções de Agrupamento

Funções de agregação como `COUNT`, `SUM`, `AVG`, `MAX` e `MIN` são usadas para resumir dados. O `GROUP BY` agrupa linhas para que a função de agregação seja aplicada a cada grupo, e o `HAVING` filtra os grupos.

Listing 3.40: SELECT – GROUP BY + COUNT: Contagem de funcionários por cargo

```
1    SELECT
2         c.nome_cargo,
3         COUNT(f.id_funcionario) AS total_funcionarios
4    FROM tb_funcionario f
5    INNER JOIN tb_cargo c ON f.id_cargo = c.id_cargo
6    GROUP BY c.id_cargo, c.nome_cargo;
```

Listing 3.41: SELECT – GROUP BY + SUM: Valor total em estoque por produto

```
1    SELECT
2         p.nome_produto,
3         SUM(e.quantidade * e.preco_estoque) AS valor_total_estoque
4    FROM tb_estoque e
5    INNER JOIN tb_produto p ON e.id_produto = p.id_produto
6    GROUP BY p.id_produto, p.nome_produto;
```

Listing 3.42: SELECT – GROUP BY + HAVING: Clientes com total gasto > R$ 1000

```
1    SELECT
2         c.nome_cliente,
3         SUM(pv.quantidade * pv.valor) AS total_gasto
4    FROM tb_cliente c
5    JOIN tb_venda v ON c.id_cliente = v.id_cliente
6    JOIN tb_produto_venda pv ON v.id_venda = pv.id_venda
7    GROUP BY c.id_cliente, c.nome_cliente
8    HAVING total_gasto > 1000.00;
```

3.7 `SELECT` nos Comandos

A instrução `SELECT` pode ser aninhada dentro de comandos de Data Manipulation Language (DML) – `INSERT`, `UPDATE` e `DELETE` – para fornecer valores ou critérios de forma dinâmica.

Listing 3.43: INSERT ... SELECT – Conceder bônus (receita extra) para todos os Vendedores

```
1
```

```
    INSERT INTO tb_receita_extra (descricao_receita, valor,
       data_recebimento, id_funcionario)
    SELECT
      'Bônus de desempenho geral',
      25.00,
      CURDATE(),
      id_funcionario
    FROM tb_funcionario
    WHERE id_cargo = 1;
```

Listing 3.44: UPDATE – Cancelar todas as vendas de um cliente específico

```
    UPDATE tb_venda
    SET status = 'cancelada'
    WHERE id_cliente = (
      SELECT id_cliente
      FROM tb_cliente
      WHERE nome_cliente = 'Gabriela Barros'
    );
```

Listing 3.45: DELETE – Remover todos os pagamentos associados a vendas canceladas

```
    DELETE FROM tb_pagamento_cliente
    WHERE id_venda IN (
      SELECT id_venda
      FROM tb_venda
      WHERE status = 'cancelada'
    );
```

4 ROTINAS DE OTIMIZAÇÃO NO BD

4.1 Visões

. . .

4.2 Funções

. . .

4.3 Procedimentos

. . .

4.4 Gatilhos

. . .

4.5 Eventos

. . .

5 CONTROLE DE ACESSO

O controle de acesso é dado por `GRANT` e `REVOKE`. As subseções a seguir seguem o PDF.

5.1 Criar Usuários

...

5.2 Atribuir Permissões

...

5.3 Revogar Permissões

...

6 TRANSAÇÕES NO BD

Defina blocos transacionais atômicos para preservar ACID. . . .

7 FALHAS NO BD

Durante a execução de qualquer software, falhas podem ocorrer; tolerâncias e tratamentos devem ser estabelecidos.

7.1 Tolerâncias à Falhas

...

7.2 Tratamento à Falhas

...

REFERÊNCIAS

- . . .