

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

**CAIO MACEDO
FERNANDO BULIGON ANTUNES
JOSÉ SEBEN
KAÍQUE MEDEIROS LIMA**

**MEMORIAL DESCRIPTIVO
Laboratório de Banco de Dados**

**SANTA HELENA
2025/2**

**CAIO MACEDO
FERNANDO BULIGON ANTUNES
JOSÉ SEBEN
KAÍQUE MEDEIROS LIMA**

**MEMORIAL DESCRIPTIVO
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	6
1.1	Descrição do Banco de Dados	6
1.2	Modelo Entidade-Relacionamento	8
1.3	Estrutura do Trabalho	8
2	BANCO DE DADOS FÍSICO	9
2.1	Definição Física	9
2.2	Carga de Dados Iniciais	15
3	CONSULTAS DE DADOS NO BD	20
3.1	Restrições de Linha	20
3.2	Funções de Linha	23
3.3	Junções de Tabelas	24
3.4	Operadores de Conjunto	26
3.5	Sub-Consultas	27
3.6	Agregação e Funções de Agrupamento	29
3.7	SELECT nos Comandos DML	30
4	ROTINAS DE OTIMIZAÇÃO NO BD	32
4.1	Visões	32
4.2	Funções	32
4.3	Procedimentos	33
4.4	Gatilhos	35
4.5	Eventos	35
5	CONTROLE DE ACESSO	36
5.1	Criar Usuários	36
5.2	Atribuir Permissões	36
5.3	Revogar Permissões	36
6	TRANSAÇÕES NO BD	37
7	FALHAS NO BD	38
7.1	Tolerâncias à Falhas	38
7.2	Tratamento à Falhas	38
	REFERÊNCIAS	39

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.

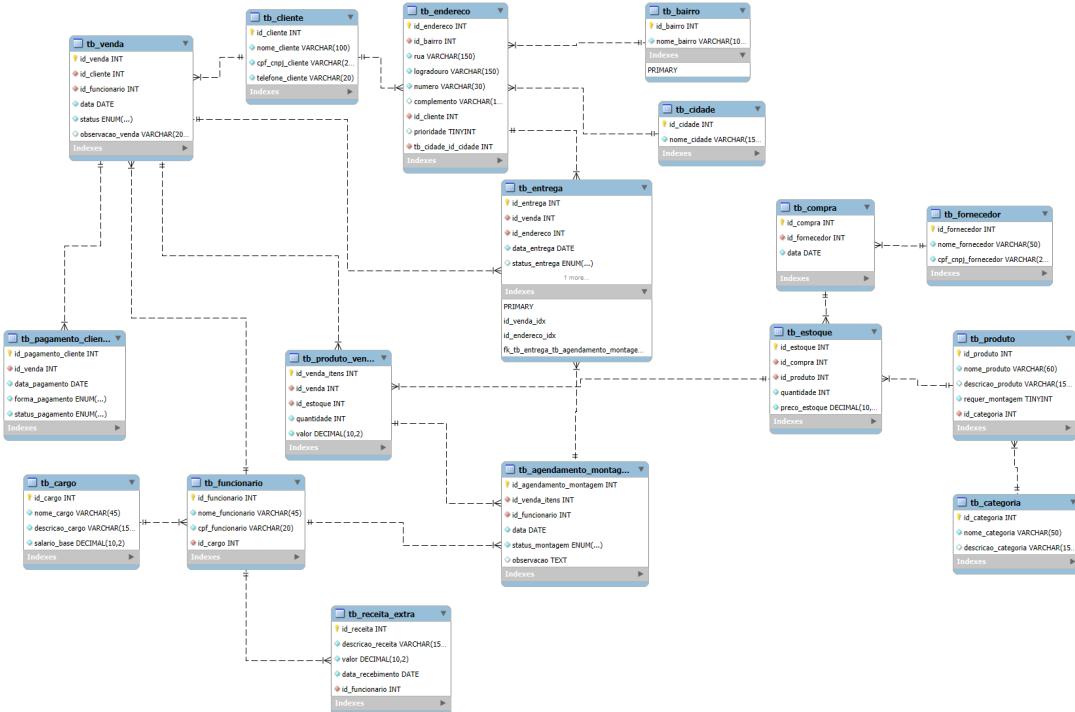


Figura 1.1: Diagrama Modelo-Entidade

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)
8     VISIBLE)
9   ENGINE = InnoDB
  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)
8     VISIBLE)
9   ENGINE = InnoDB
  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,
```

```

5   'id_cargo' INT NOT NULL ,
6   PRIMARY KEY ('id_funcionario'),
7   UNIQUE INDEX 'cpf_funcionario_UNIQUE' ('cpf_funcionario' ASC)
8       VISIBLE,
9   INDEX 'id_cargo_idx' ('id_cargo' ASC) VISIBLE,
10  CONSTRAINT 'fk_funcionario_cargo'
11  FOREIGN KEY ('id_cargo')
12  REFERENCES 'mydb'.'tb_cargo' ('id_cargo')
13  ENGINE = InnoDB
14  DEFAULT CHARACTER SET = utf8mb4;

```

Listing 2.4: DDL – Tabela tb_venda

```

1 CREATE TABLE IF NOT EXISTS 'mydb'.'tb_venda' (
2   'id_venda' INT NOT NULL AUTO_INCREMENT ,
3   'id_cliente' INT NOT NULL ,
4   'id_funcionario' INT NOT NULL ,
5   'data' DATE NOT NULL ,
6   'status' ENUM('pendente', 'concluida', 'cancelada') NOT NULL ,
7   'observacao_venda' VARCHAR(200) NULL DEFAULT NULL ,
8   PRIMARY KEY ('id_venda'),
9   INDEX 'id_cliente_idx' ('id_cliente' ASC) VISIBLE ,
10  INDEX 'id_funcionario_idx' ('id_funcionario' ASC) VISIBLE ,
11  CONSTRAINT 'fk_venda_cliente'
12  FOREIGN KEY ('id_cliente')
13  REFERENCES 'mydb'.'tb_cliente' ('id_cliente'),
14  CONSTRAINT 'fk_venda_funcionario'
15  FOREIGN KEY ('id_funcionario')
16  REFERENCES 'mydb'.'tb_funcionario' ('id_funcionario')
17  ENGINE = InnoDB
18  DEFAULT CHARACTER SET = utf8mb4;

```

Listing 2.5: DDL – Tabela tb_cidade

```

1 CREATE TABLE IF NOT EXISTS 'mydb'.'tb_cidade' (
2   'id_cidade' INT NOT NULL AUTO_INCREMENT ,
3   'nome_cidade' VARCHAR(150) NOT NULL ,
4   PRIMARY KEY ('id_cidade'),
5   UNIQUE INDEX 'nome_cidade_UNIQUE' ('nome_cidade' ASC) VISIBLE )
6   ENGINE = InnoDB
7   DEFAULT CHARACTER SET = utf8mb4;

```

Listing 2.6: DDL – Tabela tb_bairro

```

1 CREATE TABLE IF NOT EXISTS `mydb`.`tb_bairro` (
2   `id_bairro` INT NOT NULL AUTO_INCREMENT,
3   `nome_bairro` VARCHAR(100) NOT NULL,
4   `id_cidade` INT NOT NULL,
5   PRIMARY KEY (`id_bairro`),
6   INDEX `id_cidade_idx`(`id_cidade` ASC) VISIBLE,
7   CONSTRAINT `fk_bairro_cidade`
8     FOREIGN KEY (`id_cidade`)
9       REFERENCES `mydb`.`tb_cidade`(`id_cidade`)
10    ENGINE = InnoDB
11    DEFAULT CHARACTER SET = utf8mb4;

```

Listing 2.7: DDL – Tabela tb_endereco

```

1 CREATE TABLE IF NOT EXISTS `mydb`.`tb_endereco` (
2   `id_endereco` INT NOT NULL AUTO_INCREMENT,
3   `id_bairro` INT NOT NULL,
4   `logradouro` VARCHAR(150) NOT NULL,
5   `numero` VARCHAR(30) NOT NULL,
6   `complemento` VARCHAR(150) NULL DEFAULT NULL,
7   `id_cliente` INT NOT NULL,
8   `prioridade` TINYINT NULL DEFAULT NULL,
9   PRIMARY KEY (`id_endereco`),
10  INDEX `id_cliente_idx`(`id_cliente` ASC) VISIBLE,
11  INDEX `id_bairro_idx`(`id_bairro` ASC) VISIBLE,
12  CONSTRAINT `fk_endereco_cliente`
13    FOREIGN KEY (`id_cliente`)
14      REFERENCES `mydb`.`tb_cliente`(`id_cliente`),
15  CONSTRAINT `fk_endereco_bairro`
16    FOREIGN KEY (`id_bairro`)
17      REFERENCES `mydb`.`tb_bairro`(`id_bairro`)
18    ENGINE = InnoDB
19    DEFAULT CHARACTER SET = utf8mb4;

```

Listing 2.8: DDL – Tabela tb_agendamento_montagem

```

1 CREATE TABLE IF NOT EXISTS `mydb`.`tb_agendamento_montagem` (
2   `id_agendamento_montagem` INT NOT NULL AUTO_INCREMENT,
3   `id_venda_itens` INT NOT NULL,
4   `id_funcionario` INT NOT NULL,
5   `id_endereco` INT NOT NULL,
6   `data` DATE NOT NULL,
7   `status_montagem` ENUM('realizado', 'pendente', 'cancelado')

```

```

    NOT NULL ,
8   `observacao` TEXT NULL DEFAULT NULL ,
9   PRIMARY KEY (`id_agendamento_montagem`),
10  INDEX `id_venda_itens_idx`(`id_venda_itens` ASC) VISIBLE ,
11  INDEX `id_funcionario_idx`(`id_funcionario` ASC) VISIBLE ,
12  INDEX `id_endereco_idx`(`id_endereco` ASC) VISIBLE ,
13  CONSTRAINT `fk_agendamento_produto_venda`
14  FOREIGN KEY (`id_venda_itens`)
15  REFERENCES `mydb`.`tb_produto_venda`(`id_venda_itens`),
16  CONSTRAINT `fk_agendamento_funcionario`
17  FOREIGN KEY (`id_funcionario`)
18  REFERENCES `mydb`.`tb_funcionario`(`id_funcionario`),
19  CONSTRAINT `fk_agendamento_endereco`
20  FOREIGN KEY (`id_endereco`)
21  REFERENCES `mydb`.`tb_endereco`(`id_endereco`)
22  ENGINE = InnoDB
23  DEFAULT CHARACTER SET = utf8mb4;

```

Listing 2.9: DDL – Tabela tb_categoria

```

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

```

Listing 2.10: DDL – Tabela tb_fornecedor

```

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

```

Listing 2.11: DDL – Tabela tb_compra

```

1 CREATE TABLE IF NOT EXISTS `mydb`.`tb_compra` (
2   `id_compra` INT NOT NULL AUTO_INCREMENT,
3   `id_fornecedor` INT NOT NULL,
4   `data` DATE NOT NULL,
5   PRIMARY KEY (`id_compra`),
6   INDEX `id_fornecedor_idx` (`id_fornecedor` ASC) VISIBLE,
7   CONSTRAINT `fk_compra_fornecedor`
8     FOREIGN KEY (`id_fornecedor`)
9       REFERENCES `mydb`.`tb_fornecedor` (`id_fornecedor`)
10    ENGINE = InnoDB
11    DEFAULT CHARACTER SET = utf8mb4;

```

Listing 2.12: DDL – Tabela tb_entrega

```

1 CREATE TABLE IF NOT EXISTS `mydb`.`tb_entrega` (
2   `id_entrega` INT NOT NULL AUTO_INCREMENT,
3   `id_venda` INT NOT NULL,
4   `id_endereco` INT NOT NULL,
5   `data_entrega` DATE NOT NULL,
6   `status_entrega` ENUM('pendente', 'entregue', 'cancelado') NULL
7     DEFAULT 'pendente',
8   PRIMARY KEY (`id_entrega`),
9   INDEX `id_venda_idx` (`id_venda` ASC) VISIBLE,
10  INDEX `id_endereco_idx` (`id_endereco` ASC) VISIBLE,
11  CONSTRAINT `fk_entrega_venda`
12    FOREIGN KEY (`id_venda`)
13      REFERENCES `mydb`.`tb_venda` (`id_venda`),
14  CONSTRAINT `fk_entrega_endereco`
15    FOREIGN KEY (`id_endereco`)
16      REFERENCES `mydb`.`tb_endereco` (`id_endereco`)
17    ENGINE = InnoDB
18    DEFAULT CHARACTER SET = utf8mb4;

```

Listing 2.13: DDL – Tabela tb_produto

```

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

```

```

9      ,
10     INDEX `id_categoria_idx`(`id_categoria` ASC) VISIBLE,
11     CONSTRAINT `fk_produto_categoria`
12       FOREIGN KEY (`id_categoria`)
13         REFERENCES `mydb`.`tb_categoria`(`id_categoria`)
14       ENGINE = InnoDB
15     DEFAULT CHARACTER SET = utf8mb4;

```

Listing 2.14: DDL – Tabela tb_estoque

```

1 CREATE TABLE IF NOT EXISTS `mydb`.`tb_estoque` (
2   `id_estoque` INT NOT NULL AUTO_INCREMENT,
3   `id_compra` INT NOT NULL,
4   `id_produto` INT NOT NULL,
5   `quantidade` INT NOT NULL,
6   `preco_estoque` DECIMAL(10,2) NOT NULL,
7   PRIMARY KEY (`id_estoque`),
8   INDEX `id_compra_idx`(`id_compra` ASC) VISIBLE,
9   INDEX `id_produto_idx`(`id_produto` ASC) VISIBLE,
10  CONSTRAINT `fk_estoque_compra`
11    FOREIGN KEY (`id_compra`)
12      REFERENCES `mydb`.`tb_compra`(`id_compra`),
13    CONSTRAINT `fk_estoque_produto`
14      FOREIGN KEY (`id_produto`)
15      REFERENCES `mydb`.`tb_produto`(`id_produto`)
16    ENGINE = InnoDB
17    DEFAULT CHARACTER SET = utf8mb4;

```

Listing 2.15: DDL – Tabela tb_pagamento_cliente

```

1 CREATE TABLE IF NOT EXISTS `mydb`.`tb_pagamento_cliente` (
2   `id_pagamento_cliente` INT NOT NULL AUTO_INCREMENT,
3   `id_venda` INT NOT NULL,
4   `data_pagamento` DATE NOT NULL,
5   `forma_pagamento` ENUM('cartao_credito', 'cartao_debito', 'pix',
6                         , 'boleto', 'dinheiro') NOT NULL,
7   `status_pagamento` ENUM('pendente', ' pago', 'cancelado', ,
8                           'atrasado') NOT NULL,
9   PRIMARY KEY (`id_pagamento_cliente`),
10  INDEX `id_venda_idx`(`id_venda` ASC) VISIBLE,
11  CONSTRAINT `fk_pagamento_venda`
12    FOREIGN KEY (`id_venda`)
13      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

```

1  INSERT INTO tb_cargo (id_cargo, nome_cargo, descricao_cargo,
2    salario_base) VALUES
3    (1, 'Vendedor', 'Responsável pelo atendimento e vendas em loja',
4      2000.00),
5    (2, 'Montador', 'Realiza montagem de móveis', 2200.00),
6    (3, 'Gerente', 'Gerencia a equipe e suas metas da loja', 4500.00);

```

Listing 2.19: DML – Tabela tb_funcionario

```

1  INSERT INTO tb_funcionario (id_funcionario, nome_funcionario,
2    cpf_funcionario, id_cargo) VALUES
3    (1, 'José Seben', '00000000000', 3),
4    (2, 'Caio Macedo', '11111111111', 1),
5    (3, 'Kaique Lima', '22222222222', 2),
6    (4, 'Fernando Buligon', '33333333333', 1);

```

Listing 2.20: DML – Tabela tb_cliente

```

1  INSERT INTO tb_cliente (id_cliente, nome_cliente,
2    cpf_cnpj_cliente, telefone_cliente) VALUES
3    (1, 'João Oliveira', '00000000000', '(00) 00000-0000'),
4    (2, 'Maria Santos', '11111111111', '(11) 11111-1111'),
5    (3, 'Eduardo Costa', '22222222222', '(22) 22222-2222'),
6    (4, 'Gabriela Barros', '33333333333', '(33) 33333-3333'),
7    (5, 'Pedro Cachoeira', '44444444444', '(44) 44444-4444');

```

Listing 2.21: DML – Tabela tb_cidade

```

1  INSERT INTO tb_cidade (id_cidade, nome_cidade) VALUES
2    (1, 'Santa Helena'),
3    (2, 'Sao Clemente'),
4    (3, 'Medianeira'),
5    (4, 'Subsede'),
6    (5, 'Entre Rios');

```

Listing 2.22: DML – Tabela tb_bairro

```

1  INSERT INTO tb_bairro (id_bairro, nome_bairro, id_cidade)
2    VALUES
3    (1, 'Centro', 1),
4    (2, 'Centro', 2),
5    (3, 'Centro', 3),
6    (4, 'Centro', 4),
7    (5, 'Centro', 5);

```

Listing 2.23: DML – Tabela tb_endereco

```
1  INSERT INTO tb_endereco (id_endereco, id_bairro, logradouro,
2      numero, complemento, id_cliente, prioridade) VALUES
3      (1,1,'Rua Xaxim','123',NULL,1,1),
4      (2,2,'Av. Brasil','321',NULL,2,1),
5      (3,3,'Rua Aroeira','231',NULL,3,1),
6      (4,4,'Rua Argentina','213',NULL,4,1),
7      (5,4,'Rua Argentina','312',NULL,5,1);
```

Listing 2.24: DML – Tabela tb_categoria

```
1  INSERT INTO tb_categoria (id_categoria, nome_categoria,
2      descricao_categoria) VALUES
3      (1,'Ferramentas','Makita, martelo, pá...'),
4      (2,'Mobilia','Guarda-roupa, mesa, bancada...'),
5      (3,'Materiais','Areia, pedra, prego...');
```

Listing 2.25: DML – Tabela tb_fornecedor

```
1  INSERT INTO tb_fornecedor (id_fornecedor, nome_fornecedor,
2      cpf_cnpj_fornecedor) VALUES
3      (1,'Kaique Madeiras','2222222222222222'),
4      (2,'João das Areias','33333333333333'),
5      (3,'Pedro Pedradas','4444444444444444'),
6      (4,'Rafael Pregos','55555555555555'),
7      (5,'Raul Rolamentos','66666666666666');
```

Listing 2.26: DML – Tabela tb_produto

```
1  INSERT INTO tb_produto (id_produto, nome_produto,
2      descricao_produto, requer_montagem, id_categoria) VALUES
3      (1,'Makita 5007N','Serra Circular 7-1/4 1800W',0,1),
4      (2,'Madeira Pinus','Madeira boa',0,3),
5      (3,'Mesa Retangular','Mesa de madeira',1,2),
6      (4,'Pedra Brita','Essa é da boa',0,3),
7      (5,'Areia Média','Funciona bem',0,3);
```

Listing 2.27: DML – Tabela tb_compra

```
1  INSERT INTO tb_compra (id_compra, id_fornecedor, data) VALUES
2      (1,1,'2025-08-10'),
3      (2,1,'2025-08-11'),
4      (3,1,'2025-08-12'),
5      (4,1,'2025-08-13');
```

Listing 2.28: DML – Tabela tb_estoque

```
1  INSERT INTO tb_estoque (id_estoque, id_compra, id_produto,
2      quantidade, preco_estoque) VALUES
3      (1,1,1, 50, 793.78),
4      (2,2,2,100, 120.00),
5      (3,2,3, 20, 450.00),
6      (4,3,4,200, 35.00),
7      (5,4,5,180, 30.00);
```

Listing 2.29: DML – Tabela tb_venda

```
1  INSERT INTO tb_venda (id_venda, id_cliente, id_funcionario,
2      data, status, observacao_venda) VALUES
3      (1,1,2,'2025-08-15','concluida','Carasaiufeliz'),
4      (2,2,2,'2025-08-16','pendente','Aguardandopagamento'),
5      (3,3,4,'2025-08-17','concluida','Sucesso'),
6      (4,4,4,'2025-08-18','cancelada','Cartãoñopassou');
```

Listing 2.30: DML – Tabela tb_produto_venda

```
1  INSERT INTO tb_produto_venda (id_venda_itens, id_venda,
2      id_estoque, quantidade, valor) VALUES
3      (1,1,1,189,793.78),
4      (2,2,1, 1,793.78),
5      (3,3,1, 1,793.78),
6      (4,4,1, 1,793.78);
```

Listing 2.31: DML – Tabela tb_pagamento_cliente

```
1  INSERT INTO tb_pagamento_cliente (id_pagamento_cliente,
2      id_venda, data_pagamento, forma_pagamento, status_pagamento)
3          VALUES
4      (1,1,'2025-08-15','pix',' pago'),
5      (2,2,'2025-08-17','boleto','pendente'),
6      (3,3,'2025-08-17','boleto',' pago'),
7      (4,4,'2025-08-18','cartao_credito','cancelado');
```

Listing 2.32: DML – Tabela tb_entrega

```
1  INSERT INTO tb_entrega (id_entrega, id_venda, id_endereco,
2      data_entrega, status_entrega) VALUES
3      (1,1,1,'2025-08-16','entregue'),
4      (2,2,2,'2025-08-20','pendente'),
5      (3,3,3,'2025-08-18','entregue');
```

Listing 2.33: DML – Tabela tb_agendamento_montagem

```
1  INSERT INTO tb_agendamento_montagem (id_agendamento_montagem,
2      id_venda_itens, id_funcionario, id_endereco, data,
3      status_montagem, observacao) VALUES
4      (1,1,2,1,'2025-08-16','realizado','Cliente chato'),
5      (2,2,2,2,'2025-08-21','pendente','No aguardo da chegada dos
6          materiais'),
7      (3,3,2,3,'2025-08-19','realizado','Sucesso');
```

Listing 2.34: DML – Tabela tb_receita_extra

```
1  INSERT INTO tb_receita_extra (id_receita, descricao_receita,
2      valor, data_recebimento, id_funcionario) VALUES
3      (1,'Rapaz merece',350.00,'2025-08-19',2),
4      (2,'Bateu o recorde em montagem de balcão',120.00,'2025-08-19',
5          3),
6      (3,'Comprou makita',150.00,'2025-08-20',1);
```

3 CONSULTAS DE DADOS NO BD

A linguagem SQL permite diferentes recursos para recuperar informações, desde filtros simples até junções complexas e subconsultas. As seções a seguir apresentam as consultas desenvolvidas para validar o banco de dados.

3.1 Restrições de Linha

A primeira consulta busca recuperar os dados de clientes específicos filtrando por uma lista de CPFs/CNPJs conhecidos.

Listing 3.1: Filtro de clientes por lista de CPF/CNPJ

```
1   SELECT * FROM tb_cliente  
2 WHERE cpf_cnpj_cliente IN ('000000000000', '111111111111');
```

Para identificar os cargos com maior remuneração, selecionamos aqueles com salário base superior a R\$ 3.000,00.

Listing 3.2: Cargos com salário superior a R\$ 3.000

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

A listagem de funcionários abaixo filtra apenas aqueles que ocupam o cargo de "Vendedor"(identificado pelo ID 1).

Listing 3.3: Listagem de Vendedores

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

Para relatório de fechamento, buscamos todas as vendas que já possuem o status de 'concluída'.

Listing 3.4: Vendas com status concluído

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

A consulta a seguir filtra cidades de uma região específica (Santa Helena e Mediâneira) para análise logística.

Listing 3.5: Filtro de Cidades da região

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

Buscamos os bairros denominados 'Centro' que pertencem especificamente às cidades de ID 4 e 5.

Listing 3.6: Bairros do centro em cidades específicas

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

Para verificar a consistência de cadastros, buscamos endereços de clientes selecionados que não possuem complemento cadastrado.

Listing 3.7: Endereços sem complemento de clientes específicos

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

Para controle de qualidade, listamos as montagens realizadas a partir do dia 19/08/2025.

Listing 3.8: Montagens realizadas recentemente

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

A consulta abaixo lista todas as categorias de produtos, exceto a categoria de 'Ferramentas'.

Listing 3.9: Categorias exceto Ferramentas

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

Utilizando o operador LIKE, buscamos fornecedores cujo CNPJ inicia com os dígitos '33'.

Listing 3.10: Busca de fornecedor por início do CNPJ

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

Para auditar as aquisições, listamos as compras feitas com o Fornecedor 1 a partir de 12/08/2025.

Listing 3.11: Compras recentes de um fornecedor específico

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

A logística precisa monitorar entregas pendentes ou que estão agendadas para datas futuras (pós 17/08/2025).

Listing 3.12: Entregas pendentes ou futuras

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

Listamos produtos da categoria 3 (Materiais) que possuem uma descrição detalhada preenchida.

Listing 3.13: Produtos da categoria 3 com descrição

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

Para gestão de inventário, buscamos lotes de estoque com grande volume (mais de 100 itens) e custo unitário baixo (até R\$ 120,00).

Listing 3.14: Itens de estoque de alto volume e baixo custo

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

O setor financeiro necessita listar pagamentos via Pix ou Boleto que já foram efetivados ('pago').

Listing 3.15: Pagamentos efetivados via Pix ou Boleto

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

Para análise de saída de produtos, verificamos itens de venda do produto 1 onde a quantidade vendida foi alta (≥ 100).

Listing 3.16: Grandes saídas do Produto 1

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

Por fim, filtramos receitas extras de alto valor (≥ 150.00) recebidas em um intervalo de datas específico.

Listing 3.17: Receitas extras altas no período

```

1   SELECT * FROM tb_receita_extra
2   WHERE data_recebimento BETWEEN '2025-08-19' AND '2025-08-20'
3       ,
4   AND valor >= 150.00;

```

3.2 Funções de Linha

As funções de linha permitem manipular dados no momento da projeção. Abaixo, convertemos o nome da categoria para maiúsculo e contamos seus caracteres.

Listing 3.18: Manipulação de Strings (UPPER e LENGTH)

```

1   SELECT id_categoria,
2       nome_categoria,
3       UPPER(nome_categoria) AS nome_maiusculo,
4       LENGTH(nome_categoria) AS tam
5   FROM tb_categoria;

```

Para apresentar o CPF formatado em relatórios, utilizamos concatenação e *substrings*.

Listing 3.19: Formatação de máscara de CPF

```

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

```

A função COALESCE é utilizada para garantir que o endereço seja exibido corretamente mesmo quando o complemento é nulo.

Listing 3.20: Concatenação segura de Endereço

```

1   SELECT id_endereco,
2       CONCAT(
3           logradouro, ', ', numero,
4           COALESCE(CONCAT(' - ', complemento), ' ')
5       ) AS endereco_completo
6   FROM tb_endereco;

```

Manipulações de data são essenciais. Abaixo formatamos a data para o padrão brasileiro e calculamos a "idade" da venda em dias.

Listing 3.21: Formatação de Data e Diferença de Dias

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

Utilizamos a estrutura condicional CASE para classificar o valor total dos itens vendidos em faixas (alto, médio, baixo).

Listing 3.22: Lógica condicional com CASE

```
1   SELECT id_venda_itens ,
2         quantidade ,
3         valor ,
4         ROUND(quantidade * valor, 2) AS total_item ,
5         CASE
6             WHEN quantidade * valor >= 1000 THEN 'alto'
7             WHEN quantidade * valor >= 100  THEN 'médio'
8             ELSE 'baixo'
9         END AS faixa_valor
10    FROM tb_produto_venda;
```

3.3 Junções de Tabelas

As junções permitem relacionar dados distribuídos. A consulta a seguir usa INNER JOIN para mostrar o nome da categoria ao lado do produto.

Listing 3.23: Junção Produto e Categoria

```
1   SELECT p.id_produto ,
2         p.nome_produto ,
3         c.nome_categoria
4   FROM tb_produto p
5   INNER JOIN tb_categoria c
6     ON c.id_categoria = p.id_categoria;
```

Para calcular o total de uma venda e saber quem comprou, realizamos junções em cadeia entre Venda, Cliente e Itens da Venda.

Listing 3.24: Total de Venda por Cliente

```
1   SELECT v.id_venda ,
2         cl.nome_cliente ,
```

```

3           SUM(pv.quantidade * pv.valor) AS total_venda
4
5   FROM tb_venda v
6
7   INNER JOIN tb_cliente cl
8       ON cl.id_cliente = v.id_cliente
9
10  INNER JOIN tb_produto_venda pv
11      ON pv.id_venda = v.id_venda
12
13  GROUP BY v.id_venda, cl.nome_cliente;

```

Utilizamos LEFT JOIN para listar todas as compras, trazendo dados do estoque quando existirem (preservando compras que ainda não deram entrada no estoque, se houver).

Listing 3.25: Compras e entrada no Estoque

```

1   SELECT co.id_compra ,
2         f.nome_fornecedor ,
3         e.id_estoque ,
4         e.quantidade
5
6   FROM tb_compra co
7
8   INNER JOIN tb_fornecedor f
9       ON f.id_fornecedor = co.id_fornecedor
10
11  LEFT JOIN tb_estoque e
12      ON e.id_compra = co.id_compra
13
14  ORDER BY co.id_compra;

```

O conceito de *Anti-Join* é aplicado para descobrir cidades cadastradas que ainda não possuem nenhum bairro vinculado.

Listing 3.26: Cidades sem bairros cadastrados

```

1   SELECT ci.id_cidade ,
2         ci.nome_cidade
3
4   FROM tb_cidade ci
5
6   LEFT JOIN tb_bairro b
7       ON b.id_cidade = ci.id_cidade
8
9   WHERE b.id_bairro IS NULL;

```

Uma SELF JOIN é utilizada na tabela de funcionários para encontrar pares de colegas que possuem o mesmo cargo.

Listing 3.27: Pares de funcionários do mesmo cargo

```

1   SELECT f1.nome_funcionario AS funcionario_1 ,
2         f2.nome_funcionario AS funcionario_2 ,
3         c.nome_cargo
4
5   FROM tb_funcionario f1

```

```

5      INNER JOIN tb_funcionario f2
6          ON f1.id_cargo = f2.id_cargo
7          AND f1.id_funcionario < f2.id_funcionario
8      INNER JOIN tb_cargo c
9          ON c.id_cargo = f1.id_cargo;

```

3.4 Operadores de Conjunto

O operador UNION é utilizado para criar uma lista única contendo nomes de clientes e fornecedores, removendo duplicatas.

Listing 3.28: União de Clientes e Fornecedores

```

1     SELECT nome_cliente AS nome, 'cliente' AS origem
2         FROM tb_cliente
3
4     UNION
5
6     SELECT nome_fornecedor AS nome, 'fornecedor' AS origem
7         FROM tb_fornecedor;

```

Diferente do anterior, o UNION ALL mantém registros duplicados, útil para contagens totais de entidades.

Listing 3.29: União total (com duplicatas)

```

1     SELECT nome_cliente AS nome
2         FROM tb_cliente
3
4     UNION ALL
5
6     SELECT nome_fornecedor AS nome
7         FROM tb_fornecedor;

```

Simulamos um INTERSECT utilizando EXISTS para encontrar nomes que figuram tanto como clientes quanto como fornecedores.

Listing 3.30: Interseção de nomes (Cliente e Fornecedor)

```

1     SELECT c.nome_cliente AS nome
2         FROM tb_cliente c
3
4     WHERE EXISTS (
5
6         SELECT 1
7             FROM tb_fornecedor f
8             WHERE f.nome_fornecedor = c.nome_cliente
9     );

```

Simulamos o operador EXCEPT (ou MINUS) para listar clientes que estão cadastrados, mas que nunca realizaram uma compra (Venda).

Listing 3.31: Clientes sem compras (Except)

```

1   SELECT c.id_cliente, c.nome_cliente
2     FROM tb_cliente c
3   WHERE NOT EXISTS (
4       SELECT 1
5         FROM tb_venda v
6        WHERE v.id_cliente = c.id_cliente
7    );

```

Por fim, unimos cidades e bairros em uma única coluna para listagem geográfica geral.

Listing 3.32: Lista geográfica unificada

```

1   SELECT nome_cidade AS nome, 'cidade' AS tipo
2     FROM tb_cidade
3   UNION
4   SELECT nome_bairro AS nome, 'bairro' AS tipo
5     FROM tb_bairro;

```

3.5 Sub-Consultas

Subconsultas escalares retornam um único valor. Abaixo, buscamos cargos que pagam o teto salarial da empresa (o maior salário base encontrado).

Listing 3.33: Cargos com o maior salário base

```

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

```

Utilizamos IN com subconsulta para obter os dados dos clientes que possuem pelo menos uma venda com status 'concluída'.

Listing 3.34: Clientes com vendas concluídas

```

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    );

```

A subconsulta correlacionada com EXISTS filtra os fornecedores para exibir apenas aqueles com quem já realizamos compras.

Listing 3.35: Fornecedores ativos em 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   );
```

Podemos filtrar vendas com base em um cálculo agregado na subconsulta. Aqui, buscamos vendas cujo valor total exceda R\$ 1.000,00.

Listing 3.36: Vendas de alto valor (Correlacionada)

```
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;
```

Utilizando uma tabela derivada (subconsulta no `FROM`), calculamos as 5 categorias com maior volume de itens vendidos.

Listing 3.37: Top 5 Categorias por volume de vendas

```
1   SELECT d.id_categoria, d.nome_categoria, d.qtd_itens
2     FROM (
3       SELECT pr.id_categoria,
4             ca.nome_categoria,
5             SUM(pv.quantidade) AS qtd_itens
6       FROM tb_produto_venda pv
7       JOIN tb_estoque es ON es.id_estoque = pv.id_estoque
8       JOIN tb_produto pr ON pr.id_produto = es.id_produto
9       JOIN tb_categoria ca ON ca.id_categoria = pr.
10          id_categoria
11      GROUP BY pr.id_categoria, ca.nome_categoria
12    ) AS d
13    ORDER BY d.qtd_itens DESC
14    LIMIT 5;
```

Com `NOT EXISTS`, identificamos produtos que exigem montagem mas que, curiosamente, nunca foram vendidos (não constam em itens de venda).

Listing 3.38: Produtos com montagem sem saída

```
1   SELECT p.id_produto, p.nome_produto
2     FROM tb_produto p
3   WHERE p.requer_montagem = 1
4   AND NOT EXISTS (
5     SELECT 1
6       FROM tb_estoque e
7      JOIN tb_produto_venda pv ON pv.id_estoque = e.
8          id_estoque
9        WHERE e.id_produto = p.id_produto
10   );
```

Subconsultas na projeção (dentro do SELECT) permitem trazer dados agregados linha a linha. Abaixo, trazemos o total de cada venda junto aos seus dados.

Listing 3.39: Total calculado na projeção

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

3.6 Agregação e Funções de Agrupamento

Para relatórios gerenciais, contamos quantos funcionários existem em cada cargo utilizando GROUP BY.

Listing 3.40: 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;
```

Calculamos o valor financeiro imobilizado em estoque, agrupado por produto.

Listing 3.41: Valor total em estoque por produto

```
1   SELECT
2         p.nome_produto,
```

```

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

```

Utilizamos a cláusula HAVING para filtrar grupos após a agregação. Aqui, listamos apenas clientes que gastaram mais de R\$ 1.000,00 no total.

Listing 3.42: Clientes VIP (Gasto > 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 DML

É possível utilizar o resultado de um SELECT para alimentar um comando INSERT. Abaixo, criamos uma receita extra de bônus automaticamente para todos os vendedores.

Listing 3.43: Inserção em massa (Bônus para Vendedores)

```

1   INSERT INTO tb_receita_extra (descricao_receita, valor,
2                                 data_recebimento, id_funcionario)
3   SELECT
4       'Bônus de desempenho geral',
5       25.00,
6       CURDATE(),
7       id_funcionario
8   FROM tb_funcionario
      WHERE id_cargo = 1;

```

Podemos atualizar registros com base em uma subconsulta. O comando a seguir cancela todas as vendas da cliente 'Gabriela Barros'.

Listing 3.44: Update com Subconsulta

```

1   UPDATE tb_venda
2   SET status = 'cancelada'
3   WHERE id_cliente =

```

```
4     SELECT id_cliente
5     FROM tb_cliente
6     WHERE nome_cliente = 'Gabriela Barros'
7     );
```

Da mesma forma, podemos deletar registros dependentes. O comando abaixo remove pagamentos vinculados a vendas que foram canceladas.

Listing 3.45: Delete com Subconsulta

```
1   DELETE FROM tb_pagamento_cliente
2   WHERE id_venda IN (
3       SELECT id_venda
4       FROM tb_venda
5       WHERE status = 'cancelada'
6   );
```

4 ROTINAS DE OTIMIZAÇÃO NO BD

4.1 Visões

Visões são tabelas virtuais que não existem fisicamente, é visível para o usuário e apenas a sua definição é armazenada. Elas são utilizadas para evitar problemas relacionados a segurança e facilidade na consulta, permitindo que o acesso seja confidencial, simples e sem alterar a estrutura do banco de dados original. Exemplos de possíveis utilizações das visões:

Listing 4.1: VIEW – venda por funcionário

```
1 CREATE VIEW vw_vendaFuncionario
2   AS SELECT id_venda, tb_venda.id_funcionario, nome_funcionario
3     , data, status
4   FROM tb_venda
5   JOIN tb_funcionario ON tb_funcionario.id_funcionario =
6     tb_venda.id_funcionario;
7   );
```

Listing 4.2: VIEW – View para produto e sua categoria

```
1 CREATE VIEW vw_produtoCategoria(nome, descricao,
2   requer_montagem, categoria)
3   AS (SELECT nome_produto, descricao_produto, requer_montagem,
4     tb_produto.id_categoria
5   FROM tb_produto
6   JOIN tb_categoria ON tb_categoria.id_categoria = tb_produto.
7     id_categoria);
```

4.2 Funções

Uma função é um procedimento armazenado dentro do servidor que é executada sempre que chamada. Ela deve retornar um valor de retorno e é utilizada para fazer parte de uma expressão ou parâmetro. Com uma função, não é possível alterar o estado atual do banco de dados. Essa função pode ser chamada tanto por um procedimento quanto para um SELECT. Exemplos de possíveis usos de funções:

Listing 4.3: FUNCTION – Retorna o faturamento com vendas no mês

```
1 DELIMITER //
2 create function VENDA_NO_MES(mes int, ano int)
3   returns decimal(10,2)
4   begin
```

```

5      declare valor_total decimal(10,2);
6
7      select sum(valor) into valor_total
8      from tb_produto_venda
9      inner join tb_venda on tb_venda.id_venda = tb_produto_venda.
10     id_venda
11    where month(tb_venda.data) = mes
12    and year(tb_venda.data) = ano;
13
14
15   end; //
16

```

4.3 Procedimentos

Procedimentos Armazenados (*Stored Procedures*) são conjuntos de comandos SQL pré-compilados e armazenados no servidor de banco de dados. Eles permitem encapsular lógica de negócios, reutilizar código, melhorar a segurança e, em muitos casos, aumentar o desempenho ao reduzir o tráfego de rede.

Abaixo são apresentados exemplos de procedimentos criados para automatizar tarefas comuns no sistema KJCF&Cia.

Listing 4.4: Procedure – Atualizar preço no estoque por percentual

```

1      DELIMITER $$

2
3      CREATE PROCEDURE pr_reajustar_preco_estoque (
4          IN p_id_produto INT,
5          IN p_percentual DECIMAL(5,2)
6      )
7      BEGIN
8          -- Atualiza o preço do produto no estoque baseado em um
9          -- percentual
10         -- Ex: 10.0 representa um aumento de 10%
11         UPDATE tb_estoque
12         SET preco_estoque = preco_estoque * (1 + (p_percentual
13             / 100))
14         WHERE id_produto = p_id_produto;
15     END $$

16
17      DELIMITER ;

```

Para chamar o procedimento acima e aplicar um aumento de 10% no produto de ID 1:

Listing 4.5: Exemplo de chamada – pr_reajustar_preco_estoque

```
1 CALL pr_reajustar_preco_estoque(1, 10.00);
```

Outro uso comum é a geração de relatórios encapsulados. O procedimento a seguir retorna o histórico de vendas de um cliente específico dentro de um período.

Listing 4.6: Procedure – Histórico de compras do cliente

```
1 DELIMITER $$

2

3 CREATE PROCEDURE pr_historico_cliente (
4     IN p_id_cliente INT,
5     IN p_data_inicio DATE,
6     IN p_data_fim DATE
7 )
8 BEGIN
9     SELECT
10         v.id_venda ,
11         v.data ,
12         v.status ,
13         SUM(pv.quantidade * pv.valor) AS valor_total
14     FROM tb_venda v
15     INNER JOIN tb_produto_venda pv ON v.id_venda = pv.
16         id_venda
17     WHERE v.id_cliente = p_id_cliente
18     AND v.data BETWEEN p_data_inicio AND p_data_fim
19     GROUP BY v.id_venda , v.data , v.status
20     ORDER BY v.data DESC;
21
22 END $$

23
24 DELIMITER ;
```

Por fim, procedimentos podem utilizar parâmetros de saída (OUT) para retornar valores calculados para a aplicação ou para outros blocos SQL.

Listing 4.7: Procedure – Calcular comissão total de funcionário (OUT)

```
1 DELIMITER $$

2

3 CREATE PROCEDURE pr_calcular_total_vendas_funcionario (
4     IN p_id_funcionario INT,
5     OUT p_total_vendido DECIMAL(10,2)
```

```

6      )
7      BEGIN
8          -- Calcula a soma total das vendas concluidas
9          -- realizadas pelo funcionario
10         SELECT SUM(pv.quantidade * pv.valor)
11         INTO p_total_vendido
12         FROM tb_venda v
13         INNER JOIN tb_produto_venda pv ON v.id_venda = pv.
14             id_venda
15         WHERE v.id_funcionario = p_id_funcionario
16         AND v.status = 'concluida';
17
18         -- Se nao houver vendas, define como 0
19         IF p_total_vendido IS NULL THEN
20             SET p_total_vendido = 0.00;
21         END IF;
22
23     END $$

24
25     DELIMITER ;

```

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

- ...