DOCUMENTAÇÃO DO SISTEMA PDV – Ponto de Venda (Baratão Jacinto Leite)

Integrantes: Eduarda Helena
Stefany Mendonça
Gustavo Martinho

1. Introdução

Este sistema é um Ponto de Venda (PDV) desenvolvido em Java com conexão a MySQL, cujo objetivo é gerenciar clientes, funcionários, produtos, vendas e gerar relatórios gerenciais.
O sistema permite que vendedores realizem vendas, enquanto gerentes têm acesso a relatórios e à gestão de pessoas e estoque.

O sistema inclui:

- Cadastro de clientes e funcionários
- Controle de estoque e produtos
- Registro de vendas, com cálculo automático de comissão
- Consultas e relatórios detalhados, com funções agregadas e análise de datas

2. Requisitos do Sistema

2.1 Requisitos Funcionais

- O sistema deve permitir cadastro, edição, consulta e exclusão de clientes e funcionários (CRUD).
- O sistema deve permitir cadastro, atualização, consulta e exclusão de produtos, fornecedores e fornecimentos.
- O sistema deve permitir que um vendedor registre uma venda, adicionando múltiplos produtos a um “carrinho” e calculando automaticamente total e comissão.
- O sistema deve atualizar automaticamente o estoque quando uma venda é realizada.

- O sistema deve gerar relatórios de vendas, comissões, produtos mais vendidos e estatísticas por dia, semana ou mês.

2.2 Requisitos Não Funcionais

- O sistema deve ser implementado em Java com JDBC.
- O banco de dados deve ser MySQL, normalizado até a 3ª forma normal (3FN).
- O sistema deve apresentar mensagens amigáveis ao usuário.
- O sistema deve permitir consultas rápidas e precisas, incluindo cálculos de comissões e ticket médio.

3. Atores do Sistema

Gerente / Administrador
Acesso total. Pode cadastrar funcionários, consultar relatórios e gerenciar estoque e produtos.

Vendedor
Realiza vendas, cadastra clientes e consulta produtos disponíveis.

4. Casos de Uso (Funcionalidades)

CSU-01: Manter Pessoas

Descrição: Permite cadastrar, atualizar, listar e excluir clientes e funcionários.

Detalhes importantes:

- Um cliente ou funcionário sempre está vinculado à tabela Pessoa.
- Ao excluir um cliente, suas vendas permanecem registradas, mas o cliente fica marcado como inativo.

CSU-02: Gerenciar Estoque

Descrição: Permite cadastrar produtos, atualizar preços, controlar quantidade em estoque e registrar fornecimentos.
Observação: Fornecimentos armazenam histórico de preço e fornecedor.

CSU-03: Realizar Venda

Descrição: O vendedor seleciona um cliente, adiciona produtos ao carrinho e finaliza a venda.

Funcionalidades automáticas:

- Cálculo do total da venda
- Cálculo da comissão do vendedor
- Atualização do estoque

CSU-04: Emitir Relatórios

Descrição: Permite gerar consultas e relatórios analíticos, como:

- Total de vendas por funcionário
- Produtos mais vendidos
- Ticket médio diário
- Dia da semana com maior movimento

5. Consultas

Consultas com Junção (JOIN)

Estas consultas cruzam dados de duas ou mais tabelas para trazer informações completas.

Arquivo: `FuncionarioDAO.java` | Método: `listarTodos`

O que faz: Recupera os dados de todos os funcionários, unindo a tabela `pessoa` (para pegar nome e CPF) com a tabela `funcionario` (para pegar salário e ID funcional).

```
SELECT p.id as pid, p.nome, p.cpf, f.id as fid, f.salario
FROM pessoa p
INNER JOIN funcionario f ON p.id = f.pessoa_id
```

Arquivo: `ClienteDAO.java` | Método: `buscarPorCpf`

O que faz: Busca um cliente específico pelo CPF, unindo `pessoa` e `cliente` para garantir que o retorno contenha os dados pessoais.

```
SELECT p.id, p.nome, p.cpf
FROM pessoa p
INNER JOIN cliente c ON p.id = c.pessoa_id
WHERE p.cpf = ?
```

Arquivo: `FuncionarioDAO.java` | Método: `listarRankingReceita`

O que faz: Realiza junções entre 4 tabelas (`funcionario`, `pessoa`, `venda`, `item_produto`) para conseguir associar o nome do funcionário ao valor dos itens que ele vendeu.

```
SELECT p.nome, SUM(ip.qtd * ip.preco_praticado) as receita_total
FROM funcionario f
INNER JOIN pessoa p ON f.pessoa_id = p.id
INNER JOIN venda v ON v.funcionario_id = f.id
INNER JOIN item_produto ip ON ip.venda_id = v.id
GROUP BY f.id
ORDER BY receita_total DESC
```

Consultas com GROUP BY e Funções Agregadas

Estas consultas realizam cálculos matemáticos no banco de dados (Soma, Contagem, Média) e agrupam os resultados.

Arquivo: `VendaDAO.java` | Método: `listarMediaVendasPorDia`

O que faz: Calcula o Ticket Médio diário. Usa `SUM` para somar o valor total dos itens vendidos e divide pelo `COUNT` de vendas distintas naquele dia, agrupando os resultados por data.

```
SELECT DATE(v.data_hora) as dia, SUM(i.qtd * i.preco_praticado) / COUNT(DISTINCT v.id)
as ticket_medio
FROM venda v
JOIN item_produto i ON v.id = i.venda_id
GROUP BY dia
ORDER BY dia DESC
```

Arquivo: `FuncionarioDAO.java` | Método: `listarRankingReceita`

O que faz: Calcula a receita total gerada por cada funcionário. Usa `SUM` para somar o total vendido e `GROUP BY` para separar os totais por ID do funcionário.

```
SELECT p.nome, SUM(ip.qtd * ip.preco_praticado) as receita_total
FROM funcionario f
INNER JOIN pessoa p ON f.pessoa_id = p.id
INNER JOIN venda v ON v.funcionario_id = f.id
INNER JOIN item_produto ip ON ip.venda_id = v.id
GROUP BY f.id
ORDER BY receita_total DESC
```

Arquivo:`VendaDAO.java` | Método: `buscarMaiorVenda`

O que faz: Identifica qual foi a venda de maior valor monetário. Soma os itens de cada venda (`SUM`), agrupa pelo ID da venda (`GROUP BY`) e ordena do maior para o menor, pegando apenas o primeiro (`LIMIT 1`).

```
SELECT v.id, SUM(i.qtd * i.preco_praticado) as total
FROM venda v
JOIN item_produto i ON v.id = i.venda_id
GROUP BY v.id
ORDER BY total DESC
LIMIT 1
```

Consultas com Funções de Datas

Estas consultas manipulam o tipo `DATETIME` do MySQL para extrair dias, meses ou anos.

Arquivo: `VendaDAO.java` | Método: `buscarPorMesAno`

O que faz: Filtra as vendas que ocorreram em um mês e ano específicos. Usa `MONTH()` e `YEAR()` para extrair essas partes da data armazenada no banco.

```
SELECT *
FROM venda
WHERE MONTH(data_hora) = ? AND YEAR(data_hora) = ?
```

Arquivo: `VendaDAO.java` | Método: `buscarDiaComMaisVendas`

O que faz:Descobre qual dia da semana (Segunda, Terça, etc.) tem mais movimento. Usa `DAYNAME()` para converter a data no nome do dia e conta quantas vendas ocorreram.

```
SELECT DAYNAME(data_hora) as dia, COUNT(*) as qtd
FROM venda
GROUP BY dia
ORDER BY qtd DESC
LIMIT 1
```

Arquivo: `VendaDAO.java` | Método: `listarMediaVendasPorDia`

O que faz: Converte o campo `data_hora` (que contém hora/minuto/segundo) apenas para a data pura (ano-mês-dia) usando a função `DATE()`, permitindo o agrupamento correto por dia.

```
SELECT DATE(v.data_hora) as dia, SUM(i.qtd * i.preco_praticado) / COUNT(DISTINCT v.id)
as ticket_medio
FROM venda v
JOIN item_produto i ON v.id = i.venda_id
GROUP BY dia
ORDER BY dia DESC
```

Consultas Aninhadas (Subqueries)

Estas são consultas dentro de outras consultas (geralmente dentro da cláusula `WHERE`).
Arquivo: `ProdutoDAO.java` | Método: `listarMaisCarosQueMedia`

O que faz: Lista produtos cujo preço é superior à média geral. A subquery `(SELECT AVG(preco_atual) FROM produto)` calcula a média primeiro para ser usada como filtro na consulta principal.

SELECT *
FROM produto
WHERE preco_atual > (SELECT AVG(preco_atual) FROM produto)

Arquivo:`ClienteDAO.java` | Método: `listarSemCompras`

O que faz: Encontra clientes inativos. A subquery seleciona todos os IDs de clientes que já compraram, e a query principal seleciona quem **NÃO** está (`NOT IN`) nessa lista.

SELECT p.id, p.nome, p.cpf
FROM pessoa p
INNER JOIN cliente c ON p.id = c.pessoa_id
WHERE c.pessoa_id NOT IN (SELECT cliente_id FROM venda WHERE cliente_id IS NOT NULL)

Arquivo: `FuncionarioDAO.java` | Método: `listarSalariosAcimaMedia`

O que faz:Filtra funcionários que ganham mais que a média salarial da empresa. Usa uma subquery para calcular a média de todos os salários e compara com o salário de cada funcionário.

SELECT p.id as pid, p.nome, p.cpf, f.id as fid, f.salario
FROM pessoa p
INNER JOIN funcionario f ON p.id = f.pessoa_id
WHERE f.salario > (SELECT AVG(salario) FROM funcionario)

6. Dicionário de Dados

Pessoa

- **id** — `INT` — **PK** — Identificador único da pessoa.
- **nome** — `VARCHAR(1024)` — Nome completo da pessoa.
- **cpf** — `CHAR(11)` — CPF sem pontuação, único.

Telefone

- **id** — `INT` — Parte da PK.
- **numero** — `VARCHAR(20)` — Número de telefone.
- **pessoa_id** — `INT` — **FK** → Pessoa(id).
- **PK composta:** (id, pessoa_id)

Cliente

- **pessoa_id** — `INT` — **PK e FK** → Pessoa(id).

Funcionário

- **id** — `INT` — **PK**
- **pessoa_id** — `INT` — **FK** → Pessoa(id)
- **salario** — `DECIMAL(10,2)` — Remuneração do funcionário

Produto

- **id** — `INT UNSIGNED` — **PK**
- **nome** — `VARCHAR(1024)`
- **preco_atual** — `DECIMAL(10,2)`
- **qtd_estoque** — `INT UNSIGNED`

Fornecedor

- **id** — `INT UNSIGNED` — **PK**
- **nome_fantasia** — `VARCHAR(1024)`
- **cnpj** — `CHAR(14)` — único
- **telefone_contato** — `VARCHAR(20)`
- **telefone_telegram** — `VARCHAR(20)` — opcional

Fornecimento

- **id** — `INT UNSIGNED` — **PK**
- **fornecedor_id** — `INT UNSIGNED` — **FK** → Fornecedor(id)
- **produto_id** — `INT UNSIGNED` — **FK** → Produto(id)
- **qtd** — `INT UNSIGNED`
- **preco_custo** — `DECIMAL(10,2) UNSIGNED`
- **data_hora** — `DATETIME`

Venda

- **id** — `INT UNSIGNED` — **PK**
- **cliente_id** — `INT` — **FK** → Cliente(pessoa_id)
- **funcionario_id** — `INT` — **FK** → Funcionario(id)
- **data_hora** — `DATETIME`
- **comissao** — `DECIMAL(3,3)` — Percentual com alta precisão

ItemProduto

- id — `INT` — PK
- venda_id — `INT UNSIGNED` — FK → Venda(id)
- produto_id — `INT UNSIGNED` — FK → Produto(id)
- qtd — `INT UNSIGNED`
- preco_praticado — `DECIMAL(10,2) UNSIGNED`

7. Modelagem Lógica (SQL)

Pessoa

```
CREATE TABLE Pessoa (

    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,

    nome VARCHAR(1024) NOT NULL,

    cpf CHAR(11) NOT NULL UNIQUE

);
```

Telefone

```
CREATE TABLE Telefone (

    id INT NOT NULL AUTO_INCREMENT,

    numero VARCHAR(20) NOT NULL,

    pessoa_id INT NOT NULL,

    PRIMARY KEY (id, pessoa_id),

    FOREIGN KEY (pessoa_id) REFERENCES Pessoa(id)

        ON DELETE CASCADE

        ON UPDATE CASCADE
```

```
);
```

Cliente

```
CREATE TABLE Cliente (

    pessoa_id INT NOT NULL PRIMARY KEY,

    FOREIGN KEY (pessoa_id) REFERENCES Pessoa(id)

        ON DELETE CASCADE

        ON UPDATE CASCADE

);
```

Funcionário

```
CREATE TABLE Funcionario (

   id INT NOT NULL AUTO_INCREMENT,

   pessoa_id INT NOT NULL,

   salario DECIMAL(10,2) NOT NULL,

   PRIMARY KEY (id, pessoa_id),

   FOREIGN KEY (pessoa_id) REFERENCES Pessoa(id)

      ON DELETE CASCADE

      ON UPDATE CASCADE

);
```

Produto

```
CREATE TABLE Produto (

   id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,

   nome VARCHAR(1024) NOT NULL,
```

```
    qtd_estoque INT UNSIGNED NOT NULL,
    preco_atual DECIMAL(10,2) NOT NULL
);
```

Fornecimento

```
CREATE TABLE Fornecimento (
    id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    fornecedor_id INT UNSIGNED,
    produto_id INT UNSIGNED,
    qtd INT UNSIGNED NOT NULL,
    preco_custo DECIMAL(10,2) UNSIGNED NOT NULL,
    data_hora DATETIME NOT NULL,
    FOREIGN KEY (fornecedor_id) REFERENCES Fornecedor(id),
    FOREIGN KEY (produto_id) REFERENCES Produto(id)
);
```

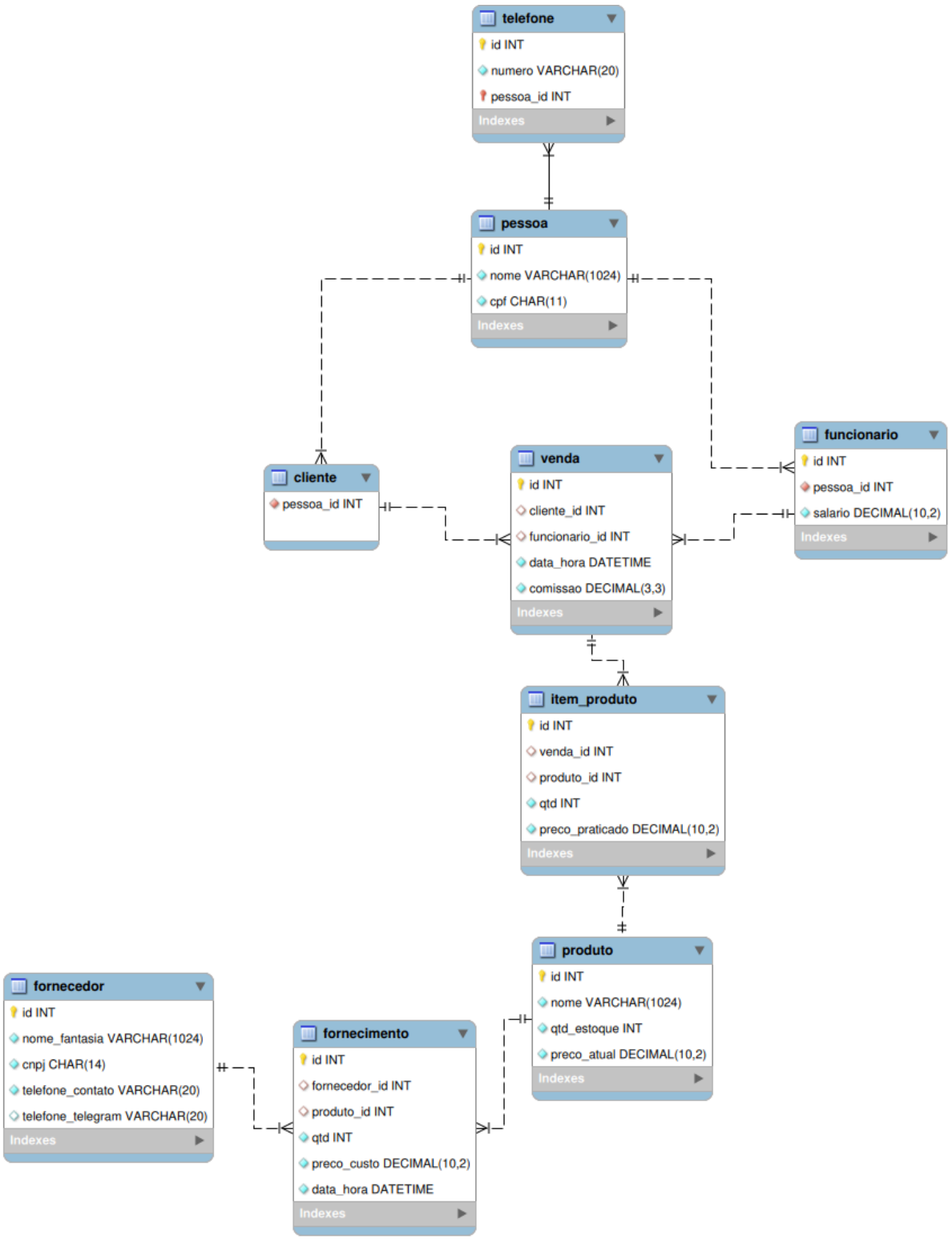
Venda

```
CREATE TABLE Venda (
    id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    cliente_id INT,
    funcionario_id INT,
    data_hora DATETIME NOT NULL,
```

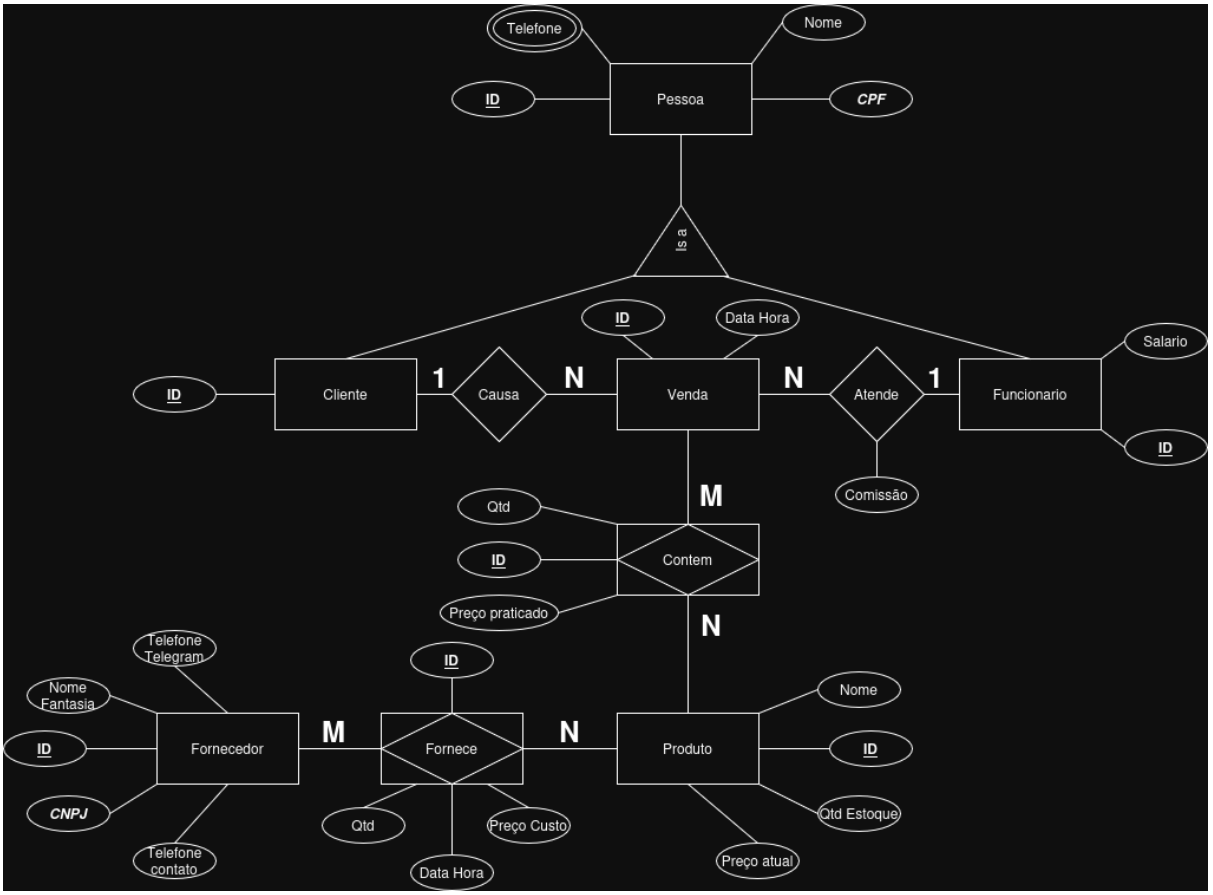
```
    comissao DECIMAL(3,3) UNSIGNED NOT NULL,
    FOREIGN KEY (cliente_id) REFERENCES Cliente(pessoa_id),
    FOREIGN KEY (funcionario_id) REFERENCES Funcionario(id)
);
```

Item Produto

```
CREATE TABLE item_produto (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    venda_id INT UNSIGNED,
    produto_id INT UNSIGNED,
    qtd INT UNSIGNED NOT NULL,
    preco_praticado DECIMAL(10,2) UNSIGNED NOT NULL,
    FOREIGN KEY (venda_id) REFERENCES Venda(id),
    FOREIGN KEY (produto_id) REFERENCES Produto(id)
);
```



8. Modelagem Conceitual (DER)



Conclusão

O desenvolvimento do sistema *Baratão Jacinto Leite* permitiu aplicar, de forma prática, conceitos de programação em Java, modelagem de banco de dados e operações CRUD. Com base nos requisitos definidos, foram elaborados os casos de uso, além da modelagem conceitual, lógica e do dicionário de dados, garantindo organização e integridade das informações.

As consultas SQL demonstraram a capacidade do banco em gerar relatórios relevantes, utilizando junções, funções agregadas, datas e consultas aninhadas.

O sistema cumpriu os objetivos propostos, resultando em um projeto completo, funcional e bem documentado, com potencial para evoluir futuramente.