□ Documentação Técnica - Sistema de Estoque Inteligente

Informações do Projeto

Projeto: Sistema de Estoque Inteligente para Microempresa de Polpas
Instituição: UNIFAMETRO - Análise e Desenvolvimento de Sistemas
SGBD: PostgreSQL 12+
Versão do Banco: 1.0
Data: Outubro 2025
Responsável: Equipe de Banco de Dados

□ Sumário

1. Visão Geral
2. Arquitetura do Banco
3. Dicionário de Dados
4. Relacionamentos
5. Regras de Negócio
6. Views e Procedures
7. Triggers e Automações
8. Índices e Performance
9. Segurança e Permissões
10. Integração com Backend
11. Exemplos de Consultas
12. Manutenção e Backup

□ Visão Geral

O banco de dados foi projetado para atender às necessidades específicas de uma microempresa de polpas de frutas, com foco em:

- ✓**Controle de estoque por lote** com rastreabilidade completa
- ✓**Alertas de vencimento** configuráveis
- ✓**Registro detalhado de vendas** com formas de pagamento
- ✓**Relatórios de perdas** para tomada de decisão
- ✓**Interface simples** para facilitar integração

✓**Automação de processos** via triggers

Principais Problemas Resolvidos

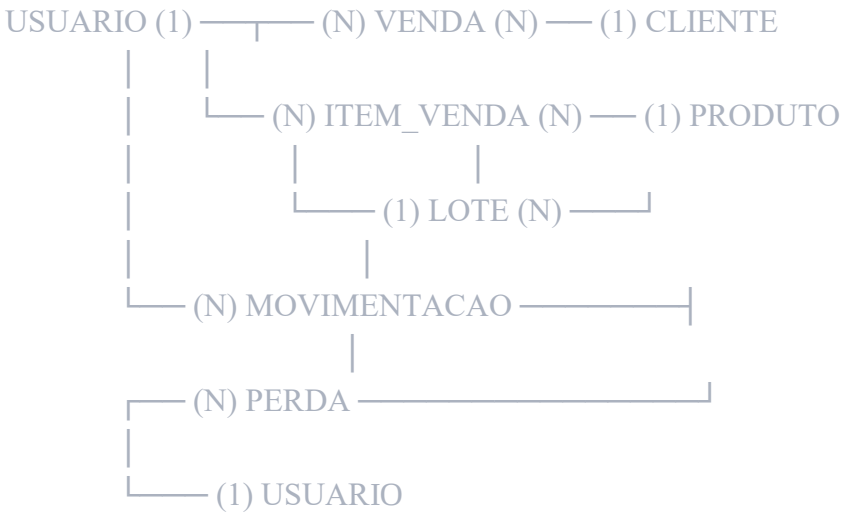
Problema	Solução Implementada
Produtos vencem sem aviso	View view_produtos_proximo_vencimento com alertas de 10 dias
Falta de controle de estoque	Tabela lote com controle por data de validade
Dificuldade em relatórios	Views pré-configuradas para relatórios rápidos

Inconsistência de dados	Triggers automáticos para atualização de estoque
Controle de perdas	Tabela perda com registro automático de vencimentos

---

Arquitetura do Banco

Diagrama de Entidades



Estrutura de Tabelas

O banco possui **8 tabelas principais**:

1. **usuario** - Controle de acesso
2. **produto** - Cadastro de polpas
3. **lote** - Controle de validade
4. **cliente** - Pedidos grandes
5. **venda** - Transações
6. **item_venda** - Detalhes das vendas
7. **movimentacao** - Histórico de estoque
8. **perda** - Registro de perdas

Dicionário de Dados

Tabela: USUARIO

Armazena os usuários do sistema com diferentes níveis de acesso.

Campo	Tipo	Restrições	Descrição
id_usuario	SERIAL	PK	Identificador único
nome	VARCHAR(100)	NOT NULL	Nome completo do usuário
email	VARCHAR(150)	UNIQUE, NOT NULL	Email de acesso
senha_hash	VARCHAR(255)	NOT NULL	Hash bcrypt da senha
perfil	VARCHAR(20)	NOT NULL, DEFAULT 'operador'	admin, gerente, operador
ativo	BOOLEAN	DEFAULT TRUE	Status do usuário

data_criacaoTIMESTAMP DEFAULT NOW() Data de cadastro

Índices:

- PK em `id_usuario`
- UNIQUE em `email`

Observações:

- Perfil `admin`: acesso total ao sistema
- Perfil `gerente`: acesso a relatórios e cadastros
- Perfil `operador`: apenas vendas e consultas

Tabela: PRODUTO

Cadastro dos produtos (polpas de frutas).

Campo	Tipo	Restrições	Descrição
id_produto	SERIAL	PK	Identificador único
nome	VARCHAR(100)	NOT NULL, UNIQUE	Nome da polpa
categoria	VARCHAR(50)	-	Tropical, Cítrica, Vermelha
unidade_medida	VARCHAR(20)	NOT NULL, DEFAULT 'kg'	kg, litro, unidade
estoque_minimo	INTEGER	DEFAULT 0, CHECK >= 0	Quantidade mínima para alerta
preco_venda	DECIMAL(10,2)	NOT NULL, CHECK > 0	Preço unitário de venda
ativo	BOOLEAN	DEFAULT TRUE	Produto ativo no sistema
data_cadastro	TIMESTAMP	DEFAULT NOW()	Data de cadastro

Índices:

- PK em `id_produto`
- INDEX em `nome` (buscas frequentes)
- INDEX em `ativo` WHERE ativo = TRUE

Observações:

- `estoque_minimo` é usado na view `view_produtos_estoque_baixo`
- Produtos inativos não aparecem em relatórios, mas mantêm histórico

---

Tabela: LOTE

Controle de lotes com data de validade (núcleo do sistema).

Campo	Tipo	Restrições	Descrição
id_lote	SERIAL	PK	Identificador único
id_produto	INTEGER	FK, NOT NULL	Referência ao produto
numero_lote	VARCHAR(50)	NOT NULL	Número do lote
data_fabricacao	DATE	-	Data de fabricação
data_validade	DATE	NOT NULL	Data de vencimento
quantidade_inicial	INTEGER	NOT NULL, CHECK > 0	Quantidade ao criar lote
quantidade_atual	INTEGER	NOT NULL, CHECK >= 0	Quantidade disponível
status	VARCHAR(20)	DEFAULT 'ativo'	ativo, vencido, esgotado
data_cadastro	TIMESTAMP	DEFAULT NOW()	Data de criação

Índices:

- PK em `id_lote`
- INDEX em `id_produto`
- INDEX em `data_validade` WHERE status = 'ativo' (consultas de vencimento)
- INDEX em `status`

Constraints:
- UNIQUE em (numero_lote, id_produto)
- CHECK: quantidade_atual <= quantidade_inicial
- CHECK: data_validade >= data_fabricacao

Observações:
- Status é atualizado automaticamente por trigger
- Lotes vencidos geram registro automático de perda

Tabela: CLIENTE

Cadastro simplificado apenas para pedidos grandes.

```
    Campo    Tipo   Restrições    Descrição id_cliente SERIAL PK
       Identificador único nome    VARCHAR(100) NOT NULL Nome do cliente
telefone        VARCHAR(20) NOT NULL, UNIQUE Telefone de contato
email          VARCHAR(150) -                   Email (opcional)
tipo_cliente VARCHAR(20) DEFAULT 'eventual' eventual, frequente data_cadastro TIMESTAMP
DEFAULT NOW() Data de cadastro
observacao     TEXT         -                   Observações gerais
```

Observações:
- Cadastro opcional, usado apenas em pedidos grandes
- Vendas pequenas não precisam de cliente

Tabela: VENDA

Registro de todas as vendas realizadas.

Campo	Tipo	Restrições	Descrição
id_venda	SERIAL	PK	Identificador único
id_usuario	INTEGER	FK, NOT NULL	Vendedor responsável
id_cliente	INTEGER	FK, NULL	Cliente (se pedido grande)
data_venda	TIMESTAMP	DEFAULT NOW()	Data/hora da venda
valor_total	DECIMAL(10,2)	NOT NULL, CHECK > 0	Valor total da venda
forma_pagamento	VARCHAR(20)	NOT NULL	dinheiro, pix, cartao_debito, cartao_credito status
	VARCHAR(20)	DEFAULT 'concluida'	concluida, cancelada observacao
	TEXT	-	Observações da venda

Índices:

- PK em `id_venda`
- INDEX em `data_venda` (relatórios por período)
- INDEX em `id_usuario`
- INDEX em `forma_pagamento`

Observações:

- `id_cliente` é NULL para vendas pequenas
- Forma de pagamento essencial para relatórios financeiros

Tabela: ITEM_VENDA

Itens individuais de cada venda com rastreamento de lote.

Campo	Tipo	Restrições	Descrição
id_item_venda	SERIAL	PK	Identificador único
id_venda	INTEGER	FK, NOT NULL	Venda relacionada
id_produto	INTEGER	FK, NOT NULL	Produto vendido
id_lote	INTEGER	FK, NOT NULL	Lote específico usado
quantidade	INTEGER	NOT NULL, CHECK > 0	Quantidade vendida
preco_unitario	DECIMAL(10,2)	NOT NULL, CHECK > 0	Preço no momento da venda subtotal
	DECIMAL(10,2)	NOT NULL, CHECK > 0	Calculado automaticamente

Índices:

- PK em `id_item_venda`
- INDEX em `id_venda`
- INDEX em `id_produto`
- INDEX em `id_lote`

Triggers:

- `trg_calcular_subtotal`: Calcula subtotal automaticamente `trg_registrar_saida_venda`:
- Cria movimentação de saída

Observações:

- Rastreabilidade total: cada item sabe de qual lote veio
- Subtotal calculado automaticamente via trigger

Tabela: MOVIMENTACAO

Histórico completo de entradas e saídas de estoque.

Campo	Tipo	Restrições	Descrição
id_movimentacao	SERIAL	PK	Identificador único
id_lote	INTEGER	FK, NOT NULL	Lote movimentado
id_usuario	INTEGER	FK, NOT NULL	Usuário responsável
tipo	VARCHAR(20)	NOT NULL	entrada, saida, ajuste
quantidade	INTEGER	NOT NULL, CHECK != 0	Positivo/negativo conforme tipo
data_movimentacao	TIMESTAMP	DEFAULT NOW()	Data/hora da movimentação observacao TEXT
- Detalhes da movimentação			

Índices:

- PK em `id_movimentacao`
- INDEX em `id_lote`
- INDEX em `data_movimentacao`
- INDEX em `tipo`

Triggers:

- `trg_atualizar_estoque`: Atualiza quantidade_atual do lote

Observações:

- Quantidade positiva para entrada, negativa para saída
- Movimentações de venda são criadas automaticamente

Tabela: PERDA

Registro de perdas para relatórios e análises.

Campo	Tipo	Restrições	Descrição
id_perda	SERIAL	PK	Identificador único
id_lote	INTEGER	FK, NOT NULL	Lote da perda
id_usuario	INTEGER	FK, NOT NULL	Usuário que registrou
motivo	VARCHAR(50)	NOT NULL	vencimento, avaria, contaminacao, outros
quantidade	INTEGER	NOT NULL, CHECK > 0	Quantidade perdida
data_perda	TIMESTAMP	DEFAULT NOW()	Data/hora da perda
observacao	TEXT	-	Detalhes da perda

Índices:

- PK em `id_perda`
- INDEX em `id_lote`
- INDEX em `data_perda`
- INDEX em `motivo`

Observações:

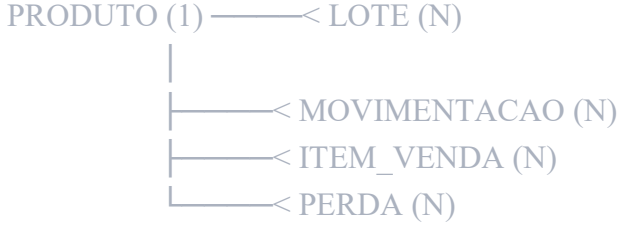
- Perdas por vencimento são registradas automaticamente
- Valor da perda calculado na view `view_relatorio_perdas`

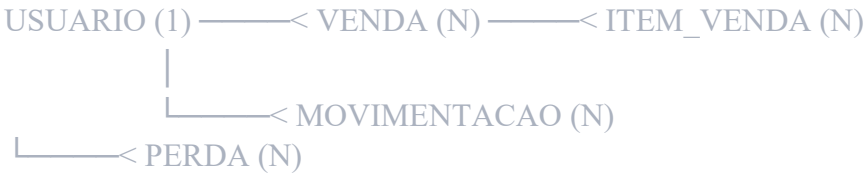
🔗 Relacionamentos

Relacionamentos 1:N (Um para Muitos)

Tabela Pai	Tabela Filha	Tipo	Descrição
produto	lote	1:N	Um produto tem vários lotes
lote	movimentacao	1:N	Um lote tem várias movimentações
lote	item_venda	1:N	Um lote fornece vários itens
lote	perda	1:N	Um lote pode ter várias perdas
usuario	venda	1:N	Um usuário realiza várias vendas
usuario	movimentacao	1:N	Um usuário faz várias movimentações
usuario	perda	1:N	Um usuário registra várias perdas
cliente	venda	1:N	Um cliente faz várias compras
venda	item_venda	1:N	Uma venda tem vários itens
produto	item_venda	1:N	Um produto está em vários itens

Cardinalidades Importantes





⚙Regras de Negócio

RN01 - Controle de Estoque Mínimo

Regra: Sistema deve alertar quando quantidade total de um produto cair abaixo do estoque mínimo.

Implementação:

- View `view_produtos_estoque_baixo`
- Exibição na tela inicial do sistema

SQL:

sql

```
SELECT * FROM view_produtos_estoque_baixo;
```

RN02 - Alerta de Vencimento

Regra: Produtos que vencem em 10 dias devem aparecer em destaque na tela inicial.

Implementação:

- View `view_produtos_proximo_vencimento`
- Margem de 10 dias configurável ajustando a view

SQL:

sql

```
SELECT * FROM view_produtos_proximo_vencimento;
```

Para alterar margem (exemplo: 7 dias):

sql

```
-- Alterar na view: CURRENT_DATE + INTERVAL '7 days'
```

RN03 - Registro Automático de Perdas

Regra: Lotes vencidos com quantidade > 0 devem gerar automaticamente registro de perda.

Implementação:

- Trigger `trg_verificar_vencimento` na tabela `lote`
- Função `fn_verificar_vencimento_lote()`

Comportamento:

1. Ao atualizar status do lote para 'vencido'
2. Se quantidade_atual > 0
3. Cria registro em `perda` com motivo 'vencimento'
4. Zera quantidade_atual do lote

RN04 - Atualização Automática de Estoque

Regra: Ao registrar uma venda, o estoque deve ser atualizado automaticamente.

Implementação:

- Trigger `trg_registrar_saida_venda` na tabela `item_venda`
- Trigger `trg_atualizar_estoque` na tabela `movimentacao`

Fluxo:

1. Inserir item_venda
2. Trigger cria movimentacao (saída)
3. Trigger atualiza quantidade_atual do lote
4. Atualiza status do lote se necessário

RN05 - Rastreabilidade de Lotes

Regra: Cada venda deve registrar de qual lote específico o produto veio.

Implementação:

- Campo `id_lote` na tabela `item_venda`
- Foreign key para tabela `lote`

Benefício:

- Rastreamento completo em caso de problemas de qualidade
- Controle preciso do FIFO (First In, First Out)

RN06 - Validação de Quantidades

Regra: Quantidade atual de um lote nunca pode ser maior que a inicial.

Implementação:

- Constraint CHECK na tabela `lote`
- `CHECK (quantidade_atual <= quantidade_inicial)`

RN07 - Formas de Pagamento

Regra: Toda venda deve ter forma de pagamento registrada.

Implementação:

- Campo obrigatório `forma_pagamento` na tabela `venda`
- CHECK constraint com valores válidos

Valores aceitos:

- dinheiro
- pix
- cartao_debito
- cartao_credito

RN08 - Cliente Opcional

Regra: Cliente só é obrigatório para pedidos grandes.

Implementação:

- Campo `id_cliente` NULL na tabela `venda`
- Decisão de cadastrar fica a critério do operador

Views e Procedures

Views Principais

1. view_produtos_estoque_baixo

Propósito: Lista produtos com estoque abaixo do mínimo.

Uso:

sql

```
SELECT * FROM view_produtos_estoque_baixo;
```

Colunas: •

- id_produto
- nome
- categoria
- estoque_minimo

estoque_total (soma de todos os lotes ativos) quantidade_faltante

Integração Backend:

javascript

```
// GET /api/dashboard/estoque-baixo
SELECT * FROM view_produtos_estoque_baixo;
```

---

2. view_produtos_proximo_vencimento

Propósito: Alerta de produtos que vencem em 10 dias.

Uso:

sql

```
SELECT * FROM view_produtos_proximo_vencimento;
```

Colunas:

- id_lote
- id_produto
- produto (nome)
- numero_lote
- data_validade
- quantidade_atual
- dias_para_vencer

---

3. view_dashboard_resumo

Propósito: Resumo geral para tela inicial.

Uso:

sql

```
SELECT * FROM view_dashboard_resumo;
```

Retorna:

- produtos_estoque_baixo
- (quantidade) produtos_vencendo
- (quantidade) vendas_hoje (valor R$)
- vendas_mes (valor R$) perdas_mes
- (quantidade)

Exemplo de resposta:

json

```
{
  "produtos_estoque_baixo": 3,
  "produtos_vencendo": 5,
  "vendas_hoje": 450.00,
  "vendas_mes": 12500.00,
  "perdas_mes": 8
}
```

---

4. view_relatorio_vendas

Propósito: Relatório detalhado de vendas.

Uso:

sql

```
SELECT * FROM view_relatorio_vendas
WHERE DATE(data_venda) = CURRENT_DATE;
```

---

5. view_relatorio_perdas

Propósito: Relatório de perdas com valor estimado.

Uso:

sql

```
SELECT * FROM view_relatorio_perdas
WHERE DATE_TRUNC('month', data_perda) = DATE_TRUNC('month', CURRENT_DATE);
```

---

6. view_produtos_mais_vendidos
Propósito: Ranking de produtos mais vendidos.

Uso:

sql

```sql
SELECT * FROM view_produtos_mais_vendidos
LIMIT 10;
```

Procedures (Functions)

1. fn_relatorio_vendas_periodo

Propósito: Relatório de vendas agregado por dia.

Uso:

sql

```sql
SELECT * FROM fn_relatorio_vendas_periodo('2025-10-01', '2025-10-31');
```

Retorna:

- data
- total_vendas (quantidade)
- valor_total (R$)
- ticket_medio (R$)

---

2. fn_relatorio_forma_pagamento

Propósito: Análise de vendas por forma de pagamento.

Uso:

sql

```sql
-- Mês atual
SELECT * FROM fn_relatorio_forma_pagamento(NULL, NULL);

-- Período específico
SELECT * FROM fn_relatorio_forma_pagamento('2025-10-01', '2025-10-31');
```

Retorna:

- forma_pagamento
- quantidade (número de vendas)
- valor_total (R$)
- percentual (%)

---

3. fn_verificar_integridade

Propósito: Verificar inconsistências nos dados.

Uso:

sql

```
SELECT * FROM fn_verificar_integridade();
```

Retorna problemas como:

- Lotes com quantidade negativa
- Vendas com valor zerado
- Lotes vencidos marcados como ativos

⚡ Triggers e Automações

Trigger 1: trg_atualizar_estoque

Tabela: movimentacao
Evento: AFTER INSERT
Função: fn_atualizar_estoque()

O que faz:

1. Atualiza `quantidade_atual` do lote
2. Atualiza `status` do lote (esgotado, vencido, ativo) **Exemplo:**

sql

```
-- Inserir entrada
INSERT INTO movimentacao (id_lote, id_usuario, tipo, quantidade, observacao)
VALUES (1, 1, 'entrada', 50, 'Compra de fornecedor');

-- Lote é atualizado automaticamente
```

Trigger 2: trg_registrar_saida_venda

Tabela: item_venda
Evento: AFTER INSERT
Função: fn_registrar_saida_venda()

O que faz: Cria automaticamente uma movimentação de saída quando um item é vendido.

Fluxo:

Venda → Item_Venda → Movimentacao (automático) → Atualiza Lote (automático)

Trigger 3: trg_calcular_subtotal

Tabela: item_venda
Evento: BEFORE INSERT OR UPDATE
Função: fn_calcular_subtotal_item()

O que faz: Calcula `subtotal = quantidade * preco_unitario` automaticamente.

Trigger 4: trg_verificar_vencimento

Tabela: lote
Evento: BEFORE UPDATE
Função: fn_verificar_vencimento_lote()

O que faz:

1. Detecta quando lote muda para status 'vencido'
2. Se tem quantidade > 0, cria registro de perda
3. Zera quantidade do lote

? Índices e Performance

Índices Criados

Tabela	Índice	Tipo	Justificativa
lote	idx_lote_validade	INDEX WHERE	Consultas de vencimento frequentes lote
	idx_lote_produto	INDEX	Join com produto venda
	idx_venda_data	INDEX	Relatórios por período venda
	idx_venda_forma_pagamento	INDEX	Análise financeira movimentacao
	idx_movimentacao_data	INDEX	Histórico temporal produto
	idx_produto_nome	INDEX	Buscas por nome

Otimizações Implementadas

1. **Índices Parciais:**
   - `idx_lote_validade WHERE status = 'ativo' idx_produto_ativo`
   - `WHERE ativo = TRUE`
2. **Índices Compostos:**
   - UNIQUE em (numero_lote, id_produto)
3. **Primary Keys:**
   - SERIAL (auto-incremento) em todas as tabelas

🔒 Segurança e Permissões

Roles para Backend Java

sql

```
-- Role principal para aplicação Spring Boot
CREATE ROLE java_app_estoque WITH LOGIN PASSWORD 'StrongP@ssw0rd2025!';

-- Permissões completas
GRANT CONNECT ON DATABASE estoque_db TO java_app_estoque;
GRANT USAGE ON SCHEMA public TO java_app_estoque;
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO java_app_estoque;
GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO java_app_estoque;
GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA public TO java_app_estoque;

-- Role somente leitura (para microserviços de relatórios)
CREATE ROLE java_app_readonly WITH LOGIN PASSWORD 'ReadOnly@2025!';
GRANT CONNECT ON DATABASE estoque_db TO java_app_readonly;
GRANT USAGE ON SCHEMA public TO java_app_readonly;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO java_app_readonly;
GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA public TO java_app_readonly;
```

Senhas e Credenciais

⚠IMPORTANTE:

- Senha padrão do admin no banco: `admin123` (ALTERAR IMEDIATAMENTE!)
- Usar BCryptPasswordEncoder do Spring Security para hash de senhas
- Armazenar credenciais do banco em variáveis de ambiente ou Azure Key Vault Nunca commitar senhas no repositório

Exemplo de uso do BCrypt:

java

```
@Service public class AuthService {    private final BCryptPasswordEncoder passwordEncoder = new
BCryptPasswordEncoder();


  public String hashPassword(String plainPassword) {       return
passwordEncoder.encode(plainPassword);

  }


  public boolean verifyPassword(String plainPassword, String hashedPassword) {       return
passwordEncoder.matches(plainPassword, hashedPassword);

  }

}
```

? Integração com Backend Java (Spring Boot)

Configuração do Projeto

1. application.properties

properties

```
# Informações da aplicação
spring.application.name=sistema-estoque-inteligente server.port=8080


# Configuração do PostgreSQL
spring.datasource.url=jdbc:postgresql://localhost:5432/estoque_db
spring.datasource.username=java_app_estoque
spring.datasource.password=StrongP@ssw0rd2025! spring.datasource.driver-class-
name=org.postgresql.Driver


# Configuração JPA/Hibernate
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto=validate spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true

spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true


# Pool de Conexões HikariCP spring.datasource.hikari.maximum-pool-size=10
spring.datasource.hikari.minimum-idle=5 spring.datasource.hikari.connection-
```

```
timeout=30000 spring.datasource.hikari.idle-timeout=600000
spring.datasource.hikari.max-lifetime=1800000

# Timezone spring.jpa.properties.hibernate.jdbc.time_zone=America/Fortaleza

# Logging
logging.level.org.hibernate.SQL=DEBUG

logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE
```

2. application-prod.properties (Produção)

properties

```
# Usar variáveis de ambiente em produção spring.datasource.url=${DB_URL}
spring.datasource.username=${DB_USERNAME}
spring.datasource.password=${DB_PASSWORD}

# Desabilitar logs em produção spring.jpa.show-
```

sql=false logging.level.org.hibernate.SQL=WARN **3.**

pom.xml - Dependências Maven

Xml:

```
<?xml version="1.0" encoding="UTF-8"?> <project
xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.2.0</version>
    <relativePath/>
  </parent>

<groupId>br.edu.unifametro</groupId>
```

```
<artifactId>sistema-estoque-inteligente</artifactId>
    <version>1.0.0</version>
<name>Sistema Estoque Inteligente</name>

    <properties>
        <java.version>17</java.version>
    </properties>

    <dependencies>
        <!-- Spring Boot Starter Web -->
        <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <!-- Spring Boot Starter Data JPA -->
        <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>

        <!-- PostgreSQL Driver -->
        <dependency>
        <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
        </dependency>

        <!-- Spring Boot Starter Validation -->
        <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-validation</artifactId>
        </dependency>

        <!-- Lombok -->
        <dependency>
        <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
```

```
<!-- Spring Boot Starter Security (para JWT) -->
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-security</artifactId>
</dependency>

<!-- JWT -->
<dependency>
<groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-api</artifactId>
    <version>0.11.5</version>
</dependency>
<dependency>
<groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-impl</artifactId>
    <version>0.11.5</version>
<scope>runtime</scope>
</dependency>
<dependency>
<groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-jackson</artifactId>
    <version>0.11.5</version>
<scope>runtime</scope>
</dependency>

<!-- Spring Boot DevTools -->
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-devtools</artifactId>
<scope>runtime</scope>
    <optional>true</optional>
</dependency>

<!-- Spring Boot Starter Test -->
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
</dependencies>
```

```
    <build>
      <plugins>
        <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
          <configuration>
            <excludes>
              <exclude>
              <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
              </exclude>
            </excludes>
          </configuration>
        </plugin>
      </plugins>
    </build>
</project>
```

Estrutura de Pacotes Sugerida

```
src/main/java/br/edu/unifametro/estoque/
├── config/
│   ├── DatabaseConfig.java
│   └── SecurityConfig.java
├── model/
│   ├── entity/
│   │   ├── Usuario.java
│   │   ├── Produto.java
│   │   ├── Lote.java
│   │   ├── Venda.java
│   │   ├── ItemVenda.java
│   │   ├── Cliente.java
│   │   ├── Movimentacao.java
│   │   └── Perda.java
```

```
|   └── dto/
|       ├── DashboardResumoDTO.java
|       ├── VendaPeriodoDTO.java
|       ├── ProdutoEstoqueBaixoDTO.java
|       └── RelatorioVendaDTO.java
├── repository/
|   ├── UsuarioRepository.java
|   ├── ProdutoRepository.java
|   ├── LoteRepository.java
|   ├── VendaRepository.java
|   └── ...
├── service/
|   ├── UsuarioService.java
|   ├── ProdutoService.java
|   ├── VendaService.java
|   ├── RelatorioService.java
|   └── DashboardService.java
├── controller/
|   ├── UsuarioController.java
|   ├── ProdutoController.java
|   ├── VendaController.java
|   ├── RelatorioController.java
|   └── DashboardController.java
└── EstoqueApplication.java
```

Exemplos de Código

1. Entidade Usuario.java

Java:

```java
java.time.LocalDateTime;

@Entity

@Table(name = "usuario")

@Data
```

```
@NoArgsConstructor @AllArgsConstructor public class Usuario {

  @Id

  @GeneratedValue(strategy = GenerationType.IDENTITY)
  @Column(name = "id_usuario")    private Long idUsuario;

  @Column(name = "nome", nullable = false, length = 100)    private String nome;

  @Column(name = "email", nullable = false, unique = true, length = 150)    private String email;

  @Column(name = "senha_hash", nullable = false, length = 255)    private String senhaHash;

  @Column(name = "perfil", nullable = false, length = 20)    private String perfil = "operador";

  @Column(name = "ativo")    private Boolean ativo = true;

  @Column(name = "data_criacao", updatable = false)    private LocalDateTime dataCriacao;

  @PrePersist    protected void onCreate() {        dataCriacao = LocalDateTime.now();    }
}
```

2. Entidade Produto.java

Java:

```
package br.edu.unifametro.estoque.model.entity;

import jakarta.persistence.*; import lombok.AllArgsConstructor; import
lombok.Data; import lombok.NoArgsConstructor; import

java.math.BigDecimal;
import java.time.LocalDateTime;

@Entity
@Table(name = "produto")
@Data
```

```
@NoArgsConstructor @AllArgsConstructor public class Produto {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "id_produto")    private Long idProduto;

   @Column(name = "nome", nullable = false, length = 100, unique = true)    private String nome;

   @Column(name = "categoria", length = 50)    private String categoria;

   @Column(name = "unidade_medida", nullable = false, length = 20)    private String unidadeMedida = "kg";

   @Column(name = "estoque_minimo")    private Integer estoqueMinimo = 0;

   @Column(name = "preco_venda", nullable = false, precision = 10, scale = 2)    private BigDecimal precoVenda;

   @Column(name = "ativo")    private Boolean ativo = true;

   @Column(name = "data_cadastro", updatable = false)    private LocalDateTime dataCadastro;

   @PrePersist    protected void onCreate() {       dataCadastro =
LocalDateTime.now();
   }
}
```

3. Entidade Lote.java

Java:

```
java.time.LocalDate;
import java.time.LocalDateTime;

@Entity
@Table(name = "lote")
@Data
@NoArgsConstructor @AllArgsConstructor public class Lote {
```

```
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_lote")    private Long idLote;

    @ManyToOne(fetch = FetchType.LAZY)    @JoinColumn(name = "id_produto", nullable = false)    private Produto produto;

    @Column(name = "numero_lote", nullable = false, length = 50)    private String numeroLote;

    @Column(name = "data_fabricacao")    private LocalDate dataFabricacao;

    @Column(name = "data_validade", nullable = false)    private LocalDate dataValidade;

    @Column(name = "quantidade_inicial", nullable = false)    private Integer quantidadeInicial;

    @Column(name = "quantidade_atual", nullable = false)    private Integer quantidadeAtual;

    @Column(name = "status", length = 20)    private String status = "ativo";

    @Column(name = "data_cadastro", updatable = false)
    private LocalDateTime dataCadastro;

    @PrePersist    protected void onCreate() {        dataCadastro =
LocalDateTime.now();
    }
}
```

4. Repository - ProdutoRepository.java

java

```
package br.edu.unifametro.estoque.repository;

import br.edu.unifametro.estoque.model.entity.Produto; import
org.springframework.data.jpa.repository.JpaRepository; import
org.springframework.data.jpa.repository.Query; import
org.springframework.stereotype.Repository; import java.util.List; import java.util.Optional;
```

```
@Repository public interface ProdutoRepository extends JpaRepository<Produto, Long> {

    // Buscar por nome
    Optional<Produto> findByNome(String nome);

    // Buscar produtos ativos
    List<Produto> findByAtivoTrue();

    // Buscar por categoria
    List<Produto> findByCategoriaAndAtivoTrue(String categoria);

    // Buscar produtos com estoque baixo usando a view
    @Query(value = "SELECT * FROM view_produtos_estoque_baixo", nativeQuery = true)    List<Object[]>
findProdutosEstoqueBaixo();

    // Contar produtos ativos
    Long countByAtivoTrue();

}

}
```

5. Repository - LoteRepository.java

java

```
package br.edu.unifametro.estoque.repository;

import br.edu.unifametro.estoque.model.entity.Lote; import
org.springframework.data.jpa.repository.JpaRepository; import
org.springframework.data.jpa.repository.Query; import
org.springframework.data.repository.query.Param; import
org.springframework.stereotype.Repository; import java.time.LocalDate; import
java.util.List;

@Repository public interface LoteRepository extends JpaRepository<Lote, Long> {

    // Buscar lotes ativos de um produto
    List<Lote> findByProdutoIdProdutoAndStatus(Long idProduto, String status);

    // Buscar lotes próximos do vencimento usando view
```

```
    @Query(value = "SELECT * FROM view_produtos_proximo_vencimento", nativeQuery = true)    List<Object[]>
  findLotesProximoVencimento();

    // Buscar lotes que vencem em determinado período
    @Query("SELECT l FROM Lote l WHERE l.status = 'ativo' " +
           "AND l.dataValidade BETWEEN :dataInicio AND :dataFim " +
           "ORDER BY l.dataValidade ASC")
    List<Lote> findLotesVencendoEntre(
    @Param("dataInicio") LocalDate dataInicio,
    @Param("dataFim") LocalDate dataFim
  );

    // Buscar lotes com estoque disponível
    @Query("SELECT l FROM Lote l WHERE l.status = 'ativo' " +
           "AND l.quantidadeAtual > 0 " +
           "ORDER BY l.dataValidade ASC")
    List<Lote> findLotesDisponiveis();
}
```

6. Service - DashboardService.java

java

```
package br.edu.unifametro.estoque.service;

import br.edu.unifametro.estoque.model.dto.DashboardResumoDTO; import jakarta.persistence.EntityManager; import
jakarta.persistence.PersistenceContext; import jakarta.persistence.Query; import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

   @Service public class DashboardService {

  @PersistenceContext    private EntityManager entityManager;

  @Transactional(readOnly = true)
    public DashboardResumoDTO getDashboardResumo() {       Query query = entityManager.createNativeQuery(
          "SELECT * FROM view_dashboard_resumo"
    );

    Object[] result = (Object[]) query.getSingleResult();
```

```
        return DashboardResumoDTO.builder()
            .produtosEstoqueBaixo(((Number) result[0]).intValue())
            .produtosVencendo(((Number) result[1]).intValue())
            .vendasHoje(((Number) result[2]).doubleValue())
            .vendasMes(((Number) result[3]).doubleValue())
        .perdasMes(((Number) result[4]).intValue())
            .build();
    }
}
```

7. Service - RelatorioService.java

java

```
package br.edu.unifametro.estoque.service;

import jakarta.persistence.EntityManager; import jakarta.persistence.PersistenceContext;
import jakarta.persistence.Query; import org.springframework.stereotype.Service; import
org.springframework.transaction.annotation.Transactional; import java.time.LocalDate;
import java.util.ArrayList; import java.util.List;

@Service
public class RelatorioService {

    @PersistenceContext     private EntityManager entityManager;

    @Transactional(readOnly = true)
    public List<Object[]> getRelatorioVendasPeriodo(LocalDate dataInicio, LocalDate dataFim) {
        Query query = entityManager.createNativeQuery(
            "SELECT * FROM fn_relatorio_vendas_periodo(:dataInicio, :dataFim)"
        );
        query.setParameter("dataInicio", dataInicio);        query.setParameter("dataFim", dataFim);

        return query.getResultList();
    }
```

```
    @Transactional(readOnly = true)
    public List<Object[]> getRelatorioFormaPagamento(LocalDate dataInicio, LocalDate dataFim) {
        String sql = "SELECT * FROM fn_relatorio_forma_pagamento(:dataInicio, :dataFim)";        Query query = entityManager.createNativeQuery(sql);        query.setParameter("dataInicio", dataInicio);        query.setParameter("dataFim", dataFim);

        return query.getResultList();
    }

    @Transactional(readOnly = true)     public List<Object[]> getRelatorioPerdas()
    {       Query query = entityManager.createNativeQuery(
            "SELECT * FROM view_relatorio_perdas ORDER BY data_perda DESC"
        );

            return query.getResultList();
    }

    @Transactional(readOnly = true)
        public List<Object[]> getProdutosMaisVendidos(Integer limite) {
        Query query = entityManager.createNativeQuery(
                "SELECT * FROM view_produtos_mais_vendidos LIMIT :limite"
        );
        query.setParameter("limite", limite);

            return query.getResultList();
    }
}
```

8. DTO - DashboardResumoDTO.java

java

```
package br.edu.unifametro.estoque.model.dto;

    import lombok.AllArgsConstructor; import lombok.Builder; import lombok.Data;
    import lombok.NoArgsConstructor;
```

@Data

@Builder

@NoArgsConstructor @AllArgsConstructor public class DashboardResumoDTO { private Integer produtosEstoqueBaixo; private Integer produtosVencendo; private Double vendasHoje; private Double vendasMes; private Integer perdasMes;

}

9. Controller - DashboardController.java

java
package br.edu.unifametro.estoque.controller;

import
br.edu.unifametro.estoque.model.dto.DashboardResumoDTO;
import br.edu.unifametro.estoque.service.DashboardService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity; import
org.springframework.web.bind.annotation.*;

@RestController

@RequestMapping("/api/dashboard")

🛠️ Manutenção e Backup

Backup Diário

```bash

# Backup completo

pg_dump -U postgres -d estoque_db -F c -b -v -f "backup_estoque_$(date +%Y%m%d).backup"

# Backup apenas dados

pg_dump -U postgres -d estoque_db -a -F p -f "backup_dados_$(date +%Y%m%d).sql"

```

Restore

```bash

# Restaurar backup completo

pg_restore -U postgres -d estoque_db -v "backup_estoque_20251015

```