Fundamentos de Modelagem Relacional e SQL

Professor: *Bernardo Petry Prates*

Kaike Torres da silva

16.11.2024

Análise e Desenvolvimento de Sistemas (ADS)

INTRODUÇÃO

Este relatório documenta o desenvolvimento de um banco de dados relacional para um sistema de vendas, focando na modelagem de entidades como clientes, fornecedores, produtos e vendas. O projeto busca consolidar habilidades em modelagem de dados, definição de chaves e aplicação de regras de negócio, garantindo a consistência e integridade dos dados. O trabalho abrange a criação de relacionamentos, restrições de integridade e procedures para automatizar operações. Ao longo do processo, desafios foram enfrentados e soluções foram adotadas, com reflexões sobre melhorias e aprendizado obtido.

RESULTADOS

1. Revisão do Negócio Escolhido

Entidades e Atributos:

1. **Clientes**
    - ClienteID (INT, PK)
    - Nome (VARCHAR)
    - Email (VARCHAR, UNIQUE)
    - Telefone (VARCHAR)
    - Endereco (VARCHAR)
    - Preferencias (TEXT)
2. **Fornecedores**
    - FornecedorID (INT, PK)
    - NomeFornecedor (VARCHAR)
    - Contato (VARCHAR)
    - Endereco (VARCHAR)
3. **Produtos**
    - ProdutoID (INT, PK)
    - NomeProduto (VARCHAR)
    - Categoria (VARCHAR)
    - Descricao (TEXT)
    - Preco (DECIMAL)

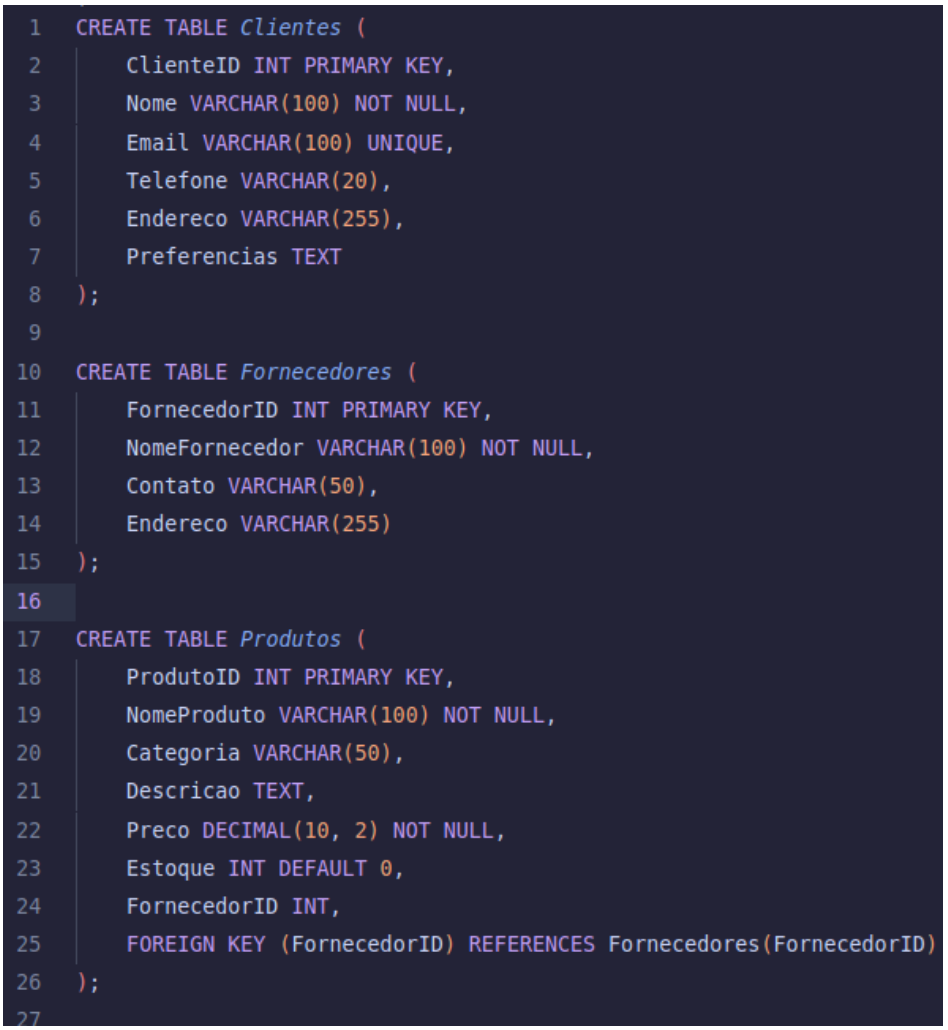
    - Estoque (INT, DEFAULT 0)
    - FornecedorID (INT, FK para Fornecedores)
4. **Vendas**
    - VendaID (INT, PK)
    - ClienteID (INT, FK para Clientes)
    - DataVenda (DATETIME, DEFAULT CURRENT_TIMESTAMP)
    - ValorTotal (DECIMAL)
    - StatusVenda (VARCHAR)
5. **ItensVenda**
    - ItemID (INT, PK)
    - VendaID (INT, FK para Vendas)
    - ProdutoID (INT, FK para Produtos)
    - Quantidade (INT)

Relacionamentos:

- Clientes - Vendas (Um-para-Muitos)
- Vendas - ItensVenda (Um-para-Muitos)
- Produtos - ItensVenda (Um-para-Muitos)
- Produtos - Fornecedores (Um-para-Muitos)

2. Criação das Tabelas

As tabelas foram criadas usando a seguinte estrutura:



```sql
CREATE TABLE Clientes (
    ClienteID INT PRIMARY KEY,
    Nome VARCHAR(100) NOT NULL,
    Email VARCHAR(100) UNIQUE,
    Telefone VARCHAR(20),
    Endereco VARCHAR(255),
    Preferencias TEXT
);

CREATE TABLE Fornecedores (
    FornecedorID INT PRIMARY KEY,
    NomeFornecedor VARCHAR(100) NOT NULL,
    Contato VARCHAR(50),
    Endereco VARCHAR(255)
);

CREATE TABLE Produtos (
    ProdutoID INT PRIMARY KEY,
    NomeProduto VARCHAR(100) NOT NULL,
    Categoria VARCHAR(50),
    Descricao TEXT,
    Preco DECIMAL(10, 2) NOT NULL,
    Estoque INT DEFAULT 0,
    FornecedorID INT,
    FOREIGN KEY (FornecedorID) REFERENCES Fornecedores(FornecedorID)
);
```

```sql
CREATE TABLE Vendas (
    VendaID INT PRIMARY KEY,
    ClienteID INT NOT NULL,
    DataVenda DATETIME DEFAULT CURRENT_TIMESTAMP,
    ValorTotal DECIMAL(10, 2),
    StatusVenda VARCHAR(20),
    FOREIGN KEY (ClienteID) REFERENCES Clientes(ClienteID)
);

CREATE TABLE ItensVenda (
    ItemID INT PRIMARY KEY,
    VendaID INT NOT NULL,
    ProdutoID INT NOT NULL,
    Quantidade INT NOT NULL,
    FOREIGN KEY (VendaID) REFERENCES Vendas(VendaID),
    FOREIGN KEY (ProdutoID) REFERENCES Produtos(ProdutoID)
);

CREATE TABLE Pagamentos (
    PagamentoID INT PRIMARY KEY,
    VendaID INT NOT NULL,
    ValorPago DECIMAL(10, 2),
    DataPagamento DATETIME DEFAULT CURRENT_TIMESTAMP,
    MetodoPagamento VARCHAR(50),
    StatusPagamento VARCHAR(20),
    FOREIGN KEY (VendaID) REFERENCES Vendas(VendaID)
);
```

```sql
CREATE TABLE Venda_Produto (
    VendaID INT,
    ProdutoID INT,
    Quantidade INT NOT NULL,
    PRIMARY KEY (VendaID, ProdutoID),
    FOREIGN KEY (VendaID) REFERENCES Vendas(VendaID),
    FOREIGN KEY (ProdutoID) REFERENCES Produtos(ProdutoID)
);
```

O padrão de nomenclatura escolhido foi no singular, e os nomes das colunas seguem o formato "CamelCase" para facilitar a leitura. As chaves primárias foram nomeadas com sufixo "ID".

3. Identificação e Definição das Chaves

Objetivo

Assegurar a integridade e unicidade dos dados, utilizando chaves primárias (PK) para garantir registros únicos e chaves estrangeiras (FK) para relacionar as tabelas de forma consistente.

Chaves Primárias (PK)

A **chave primária** é um atributo ou conjunto de atributos que identificam de maneira única cada registro em uma tabela. Para cada tabela, foi escolhida uma chave primária que garante a unicidade de cada registro.

1. **Clientes**
    - **Chave Primária**: `ClienteID`
      A chave primária `ClienteID` é única para cada cliente, garantindo que cada cliente tenha um identificador exclusivo.

```sql
CREATE TABLE Clientes (
    ClienteID INT AUTO_INCREMENT PRIMARY KEY,
    Nome VARCHAR(100),
    Email VARCHAR(100) UNIQUE,
    Telefone VARCHAR(15),
    Endereco VARCHAR(200),
    Preferencias TEXT
);
```

2. **Fornecedores**

- **Chave Primária**: `FornecedorID`
  A chave primária `FornecedorID` é única para cada fornecedor, garantindo a exclusividade de cada fornecedor.
-

```sql
CREATE TABLE Fornecedores (
    FornecedorID INT AUTO_INCREMENT PRIMARY KEY,
    NomeFornecedor VARCHAR(100),
    Contato VARCHAR(100),
    Endereco VARCHAR(200)
);
```

3. **Produtos**

- **Chave Primária**: `ProdutoID`
  A chave primária `ProdutoID` identifica exclusivamente cada produto no sistema.

```sql
CREATE TABLE Produtos (
    ProdutoID INT AUTO_INCREMENT PRIMARY KEY,
    NomeProduto VARCHAR(100),
    Categoria VARCHAR(100),
    Descricao TEXT,
    Preco DECIMAL(10, 2),
    Estoque INT DEFAULT 0,
    FornecedorID INT,
    FOREIGN KEY (FornecedorID) REFERENCES Fornecedores(FornecedorID)
);
```

4. **Vendas**

- **Chave Primária**: `VendaID`
  A chave primária `VendaID` é única para cada venda registrada no sistema.

```sql
CREATE TABLE Vendas (
    VendaID INT AUTO_INCREMENT PRIMARY KEY,
    ClienteID INT,
    DataVenda DATETIME DEFAULT CURRENT_TIMESTAMP,
    ValorTotal DECIMAL(10, 2),
    StatusVenda VARCHAR(50),
    FOREIGN KEY (ClienteID) REFERENCES Clientes(ClienteID)
);
```

5. **ItensVenda**

- **Chave Primária**: `ItemID`
  A chave primária `ItemID` é única para cada item de venda registrado, garantindo que cada item tenha um identificador exclusivo.

```sql
CREATE TABLE ItensVenda (
    ItemID INT AUTO_INCREMENT PRIMARY KEY,
    VendaID INT,
    ProdutoID INT,
    Quantidade INT,
    FOREIGN KEY (VendaID) REFERENCES Vendas(VendaID),
    FOREIGN KEY (ProdutoID) REFERENCES Produtos(ProdutoID)
);
```

Chaves Estrangeiras (FK)

As **chaves estrangeiras** são atributos em uma tabela que fazem referência à chave primária de outra tabela, criando assim o relacionamento entre as tabelas.

1. **Vendas**
    - **Chave Estrangeira**: `ClienteID`
      A coluna `ClienteID` em `Vendas` referencia a tabela `Clientes`, relacionando uma venda a um cliente específico.

```
FOREIGN KEY (ClienteID) REFERENCES Clientes(ClienteID)
```

ItensVenda

- **Chaves Estrangeiras**: `VendaID` e `ProdutoID`
  A coluna `VendaID` em `ItensVenda` referencia a tabela `Vendas`, enquanto a coluna `ProdutoID` em `ItensVenda` referencia a tabela `Produtos`, estabelecendo a relação entre itens, vendas e produtos.

```
FOREIGN KEY (VendaID) REFERENCES Vendas(VendaID),
FOREIGN KEY (ProdutoID) REFERENCES Produtos(ProdutoID)
```

Produtos

- **Chave Estrangeira**: `FornecedorID`
  A coluna `FornecedorID` em `Produtos` referencia a tabela `Fornecedores`, associando cada produto a um fornecedor.

```
FOREIGN KEY (FornecedorID) REFERENCES Fornecedores(FornecedorID)
```

4. Estabelecimento dos Relacionamentos

Objetivo

Garantir que os dados estejam corretamente interligados entre as tabelas através de **chaves estrangeiras** (FK) e estabelecer **restrições de integridade referencial**.

Relacionamentos Um-para-Muitos

- **Clientes e Vendas**: Um cliente pode fazer várias vendas, mas cada venda é associada a um único cliente.
    - **Chave estrangeira**: `ClienteID` na tabela `Vendas` referenciando `ClienteID` em `Clientes`.

```sql
ALTER TABLE Vendas
ADD CONSTRAINT FK_Vendas_Clientes
FOREIGN KEY (ClienteID) REFERENCES Clientes(ClienteID);
```

Relacionamentos Muitos-para-Muitos

- **Produtos e Vendas**: Uma venda pode ter vários produtos, e um produto pode estar em várias vendas.
    - Tabela associativa `ItensVenda` com chaves estrangeiras `VendaID` e `ProdutoID`.

```sql
CREATE TABLE ItensVenda (
    ItemID INT AUTO_INCREMENT PRIMARY KEY,
    VendaID INT,
    ProdutoID INT,
    Quantidade INT,
    FOREIGN KEY (VendaID) REFERENCES Vendas(VendaID),
    FOREIGN KEY (ProdutoID) REFERENCES Produtos(ProdutoID)
);
```

Restrições de Integridade Referencial

- **Exclusão em cascata**: Quando um **cliente** ou **produto** é excluído, as vendas ou itens associados são removidos automaticamente.

```sql
ALTER TABLE Vendas
ADD CONSTRAINT FK_Vendas_Clientes
FOREIGN KEY (ClienteID) REFERENCES Clientes(ClienteID)
ON DELETE CASCADE;
```

5. Definição das Regras de Negócio

Objetivo

Implementar as regras de negócio diretamente no banco de dados, assegurando que os dados armazenados sejam consistentes com as políticas e os processos da aplicação. Regras de negócio definidas diretamente no banco de dados garantem a segurança, integridade e consistência das informações.

Regras de Negócio Identificadas e Implementadas

1. Restrições de Integridade

Regra de Negócio 1: Todos os clientes devem ter um e-mail único.

- **Implementação**: A coluna `Email` na tabela `Clientes` possui a restrição `UNIQUE`.

```sql
CREATE TABLE Clientes (
    ClienteID INT AUTO_INCREMENT PRIMARY KEY,
    Nome VARCHAR(100) NOT NULL,
    Email VARCHAR(100) UNIQUE,
    Telefone VARCHAR(15),
    Endereco VARCHAR(200),
    Preferencias TEXT
);
```

Regra de Negócio 2: Não é permitido registrar um produto sem preço.

- **Implementação**: A coluna `Preco` na tabela `Produtos` foi definida como `NOT NULL`.

```sql
CREATE TABLE Produtos (
    ProdutoID INT AUTO_INCREMENT PRIMARY KEY,
    NomeProduto VARCHAR(100) NOT NULL,
    Categoria VARCHAR(100),
    Descricao TEXT,
    Preco DECIMAL(10, 2) NOT NULL,
    Estoque INT DEFAULT 0,
    FornecedorID INT,
    FOREIGN KEY (FornecedorID) REFERENCES Fornecedores(FornecedorID)
);
```

Regra de Negócio 3: O estoque de um produto não pode ser negativo.

- **Implementação**: Uma `CHECK constraint` foi adicionada para garantir que o valor de `Estoque` seja igual ou maior que 0.

```sql
CREATE TABLE Produtos (
    ProdutoID INT AUTO_INCREMENT PRIMARY KEY,
    NomeProduto VARCHAR(100) NOT NULL,
    Categoria VARCHAR(100),
    Descricao TEXT,
    Preco DECIMAL(10, 2) NOT NULL,
    Estoque INT DEFAULT 0 CHECK (Estoque >= 0),
    FornecedorID INT,
    FOREIGN KEY (FornecedorID) REFERENCES Fornecedores(FornecedorID)
);
```

Regra de Negócio 4: Cada venda deve estar vinculada a um cliente válido.

- **Implementação**: A coluna `ClienteID` na tabela `Vendas` é uma `FOREIGN KEY` que referencia a tabela `Clientes`.

```sql
CREATE TABLE Vendas (
    VendaID INT AUTO_INCREMENT PRIMARY KEY,
    ClienteID INT NOT NULL,
    DataVenda DATETIME DEFAULT CURRENT_TIMESTAMP,
    ValorTotal DECIMAL(10, 2),
    StatusVenda VARCHAR(50),
    FOREIGN KEY (ClienteID) REFERENCES Clientes(ClienteID)
);
```

Regras de Validação Adicional

- **Regra de Negócio 5: Um cliente não pode fazer uma venda se não houver produtos suficientes em estoque.**
    - **Implementação**: Essa regra foi implementada na **procedure** `RealizarVenda` que verifica o estoque antes de confirmar a venda. Caso o estoque seja insuficiente, um `SIGNAL` é emitido para interromper a operação.

```sql
CREATE PROCEDURE RealizarVenda(
    IN p_ClienteID INT,
    IN p_ProdutoID INT,
    IN p_Quantidade INT
)
BEGIN
    DECLARE estoque_atual INT;
    SELECT Estoque INTO estoque_atual
    FROM Produtos
    WHERE ProdutoID = p_ProdutoID;

    IF estoque_atual < p_Quantidade THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Estoque insuficiente para a venda.';
    ELSE
        -- Lógica de venda
    END IF;
END;
```

Limitações Encontradas e Alternativas

- **Limitação**: Algumas regras de validação mais complexas, como verificar o estoque antes de uma venda ou cálculos de valores, podem ser difíceis de implementar exclusivamente com `CHECK constraints` e requerem `PROCEDURES` ou lógica adicional na camada da aplicação.
- **Solução Alternativa**: Regras que envolvem cálculos, interações dinâmicas ou que são dependentes de condições complexas podem ser validadas na aplicação utilizando linguagens como JavaScript ou frameworks específicos de backend para garantir que apenas dados válidos sejam enviados ao banco.

6. Reflexão Final

Dificuldades Encontradas

Durante o processo de modelagem e implementação do banco de dados, algumas dificuldades merecem destaque:

- **Complexidade dos Relacionamentos**: A definição dos relacionamentos, especialmente ao lidar com um modelo que incluía tabelas associativas, como `ItensVenda`, exigiu um cuidado especial para garantir que todos os vínculos fossem bem definidos e funcionais. A criação e manutenção de chaves estrangeiras demandaram atenção para garantir a integridade referencial entre as entidades.
- **Escolha dos Tipos de Dados**: Encontrar o tipo de dado adequado para cada coluna foi um desafio, já que isso impacta diretamente o desempenho e a flexibilidade do banco. Por exemplo, foi necessário decidir entre `VARCHAR` e `TEXT` em campos de texto e determinar a precisão dos valores monetários com `DECIMAL`.
- **Implementação de Regras de Negócio**: A implementação de regras de negócio diretamente no banco de dados, como verificar a disponibilidade de estoque antes de confirmar uma venda, exigiu a utilização de `PROCEDURES` e `TRIGGERS`. Embora essas ferramentas ofereçam maior controle, elas também trouxeram complexidade adicional, principalmente em termos de depuração e manutenção do código.

Possíveis Otimizações de Desempenho

- **Índices Adicionais**: Para melhorar a performance das consultas, especialmente em tabelas grandes, a criação de índices adicionais em colunas frequentemente utilizadas para `JOINs` e `WHERE` pode ser uma otimização válida.
- **Normalização Adequada**: Revisar o grau de normalização pode reduzir a redundância e melhorar a consistência dos dados. Contudo, em alguns casos, a desnormalização controlada pode ser considerada para otimizar consultas específicas que exigem muitos `JOINs`.

- **Revisão de Restrições**: Reavaliar as restrições `NOT NULL` e `DEFAULT` pode garantir maior eficiência no armazenamento e consistência nas entradas, reduzindo a necessidade de validações adicionais em nível de aplicação.

Sugestões de Melhoria

- **Divisão de Tabelas e Criação de Relacionamentos Mais Complexos**: Para melhorar a flexibilidade do sistema, a criação de tabelas adicionais para rastrear logs de vendas ou alterações em produtos poderia ser uma evolução na estrutura atual.
- **Validações Avançadas na Camada de Aplicação**: Algumas regras de negócio que não foram implementadas no banco de dados devido à sua complexidade poderiam ser abordadas na camada de aplicação. Isso garantiria uma melhor experiência para o usuário e diminuiria a carga de validação diretamente no banco.
- **Documentação e Padronização de Código**: Manter uma documentação detalhada sobre as `PROCEDURES`, `TRIGGERS` e outras funções é fundamental para garantir a manutenção e a evolução do banco de dados.

Reflexão sobre o Aprendizado

Este projeto foi uma experiência enriquecedora, pois proporcionou um aprendizado profundo sobre modelagem e implementação de bancos de dados relacionais. Compreender os desafios na definição de chaves primárias e estrangeiras, a aplicação de regras de negócio e a otimização de consultas foram lições valiosas. O trabalho contribuiu para consolidar habilidades técnicas de modelagem de dados, além de desenvolver uma maior compreensão das necessidades do negócio e como traduzi-las em um sistema robusto e funcional.