BANCO DE DADOS II

PROJETO LIVRARIA

GRUPO 6

Catarina Lima Mendes 172787
Italo Victor Silva Cardoso 234499
Jullia Souza De Avelar 214539
Natalia Emboava 173709
Vinícius Lenci De Souza Aguiar 174278

1. Objetivo Geral do Projeto

Projetar um banco de dados para uma Livraria, em um sistema que permita a venda e aluguel de livros, assim como a venda de materiais escolares; registrar informações de cada compra e/ou aluguel dos produtos, assim como a dos clientes; gerenciamento do controle de produtos da livraria e de funcionários; informações em relação aos livros e materiais escolares.

2. Requisitos Não Funcionais

Em relação aos funcionários:

- O funcionário possui nome, identificação única de funcionário, CPF, cargo que ocupa, endereço, e-mail, telefone, turno em que trabalha e login no sistema;
- Um funcionário é responsável por registrar os dados cadastrais de um cliente;
- Cada funcionário pode atender diversos clientes;
- Cada funcionário pode vender ou alugar vários produtos para um cliente;
- O funcionário deve registrar as vendas e aluguéis de livros ou as vendas de materiais escolares;
- Caso o cliente compre um livro ou um material escolar, deve ser registrado a quantidade comprada;
- Caso o cliente deseje alugar um livro, deve ser registrado a data em que foi alugado e a data de devolução.

Em relação aos clientes:

- Possui identificador, nome, CPF, endereço, e-mail e número de telefone;
- Um cliente pode comprar e/ou alugar um ou mais livros;
- Um cliente pode comprar um ou mais materiais escolares;
- Cada cliente tem registrado a data de determinada compra/aluguel, o valor total pago e o código da compra.

Em relação aos atendimentos:

- Possui código que identifica o atendimento, a data em que ocorreu e o valor total resultante da venda ou do aluguel;
- Em vários atendimentos pode-se alugar vários livros;
- Em vários atendimentos pode-se vender vários produtos;

Em relação aos livros:

- Possui um identificador, preço de venda, quantidade no estoque, ISBN, título, autor, gênero, editora e preço de aluguel;

- Um livro tem um valor para aluguel e outro para venda, assim como um valor para multa caso ultrapasse a data da devolução;
- Alguns livros de mesmo título são separados para serem alugados ou vendidos.

Em relação aos materiais escolares:

- Possui um identificador, quantidade no estoque, preço unitario, modelo, marca e descrição;
- Vários materiais escolares podem ser vendidos para diversos clientes;
- Materiais escolares são vendidos por unidade, portanto, têm valor unitário.

Em relação aos fornecedores:

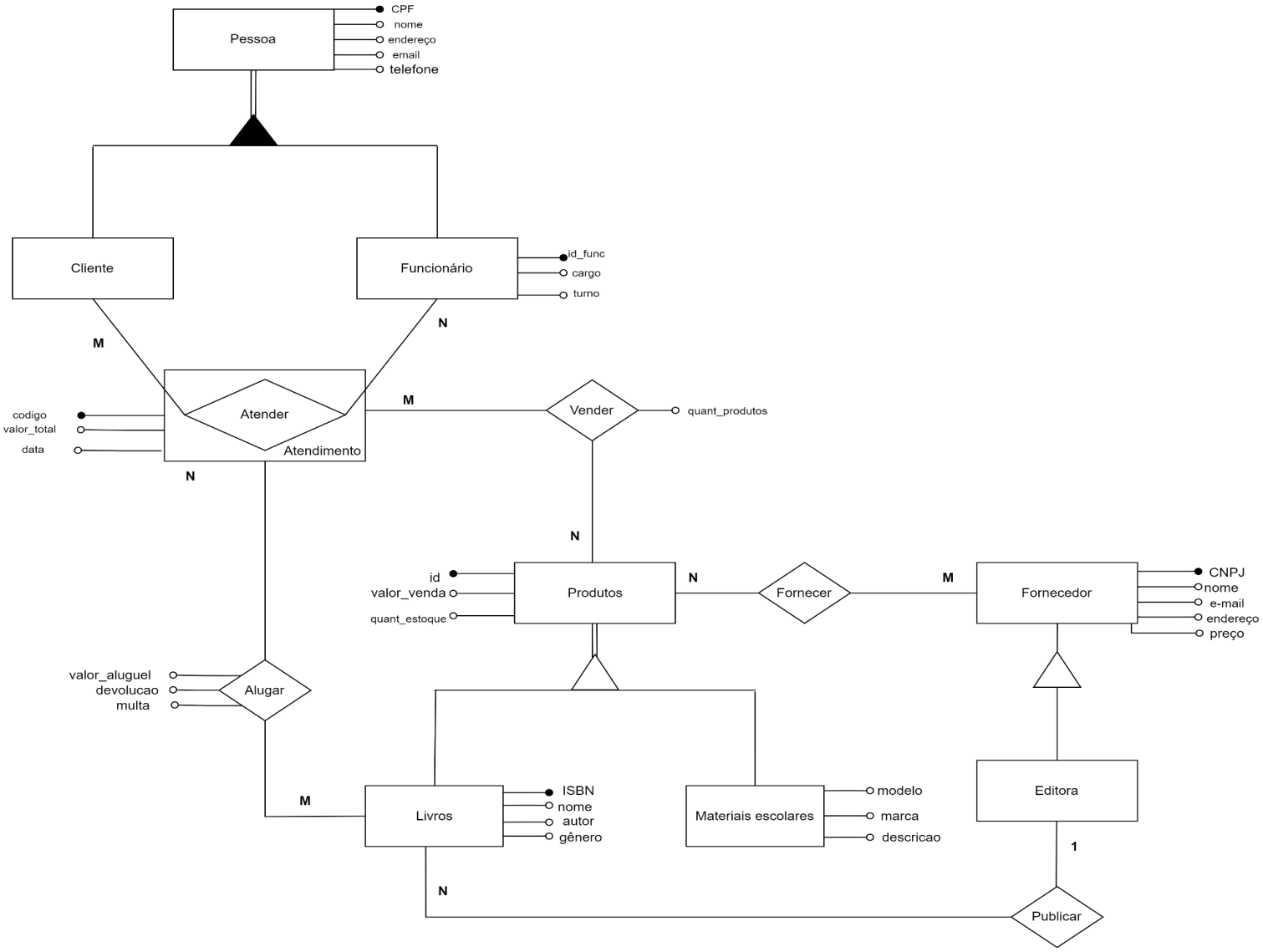
- Possui identificador, nome, e-mail, telefone, CNPJ;
- Vários materiais escolares são oferecidos por diversos fornecedores.
- Uma editora pode publicar vários livros.

3. Requisitos Funcionais

- Permitir o cadastro de clientes;
- Permitir o cadastro de funcionários;
- Permitir o cadastro de fornecedores;
- Possibilitar pagamentos seguros das vendas e aluguéis da livraria;
- Permitir registros de aluguéis de livros, a data de aluguel e a data de devolução;
- Adicionar valor de multa no aluguel caso tenha passado o dia da devolução;
- Permitir o cadastro de novos livros no sistema;
- Exibir disponibilidade do livro para alugar;
- Exibir informações sobre os livros como: autor, título, gênero;
- Permitir o cadastro de novos materiais escolares no sistema;
- Disponibilizar informações sobre os preços dos livros e materiais escolares;
- Exibir notificação para o funcionário sobre um livro em atraso de devolução.

4. Projeto Conceitual

ME-R ESTENDIDO:



5. Projeto Lógico

ESQUEMA RELACIONAL FINAL:

Pessoa = {CPF, nome, endereço, email, telefone}

Cliente = {CPF}
CPF – chave estrangeira referenciando Pessoa

Funcionário = {CPF, id_func, cargo, turno}
CPF – chave estrangeira referenciando Pessoa

Produtos = {id, valor_venda, quant_estoque, tipo}
tipo pode ser: 1 – Livros , 2 - MateriaisEscolares

Livros = {id, ISBN, nome, autor, gênero, CNPJ}
id – chave estrangeira referenciando Produtos
CNPJ - chave estrangeira de Editora

Materiais Escolares = {id, modelo, marca, descrição}
id – chave estrangeira referenciando Produtos

Fornecedor = {CNPJ, nome, email, endereço, tipo}
tipo pode ser: 1 – Editora

Editora = {CNPJ}
CNPJ – chave estrangeira referenciando Fornecedor

Atendimento = {codigo, CPF_cli, CPF_func, valor_total, data}
CPF_cli - chave estrangeira referenciando Cliente
CPF_func - chave estrangeira referenciando Funcionário

Vender = {codigo_atend, id_prod, quant_produtos}
codigo_atend - chave estrangeira referenciando Atendimento
id_prod - chave estrangeira referenciando Produtos

Alugar = {codigo_atend, id_livro, devolução, valor_aluguel, multa}
codigo_atend - chave estrangeira referenciando Atendimento
id_livro - chave estrangeira referenciando Livros

Fornecer = {id_prod, CNPJ}
id_prod – chave estrangeira referenciando Produtos
CNPJ – chave estrangeira referenciando Fornecedor

6. Projeto Físico da Base de dados

Script para a criação da base de dados no MS-SQLServer(utilizando a DDL do SQL-Server) contendo todas as restrições de integridade - chaves primárias, secundárias (se houver) e estrangeiras. Criar índices para as chaves estrangeiras.

```
USE master

GO

CREATE DATABASE livraria
GO

use livraria
CREATE TABLE pessoa (
    cpf VARCHAR(11) not null,
    nome VARCHAR(50) not null,
    endereco VARCHAR(30) not null,
    email VARCHAR(30) not null,
    telefone VARCHAR(15) not null,
    PRIMARY KEY (cpf)
)

CREATE TABLE cliente (
    cpf VARCHAR(11) not null,
    PRIMARY KEY (cpf),
    FOREIGN KEY (cpf) REFERENCES pessoa
)

CREATE TABLE funcionario (
    cpf VARCHAR(11) not null,
    id_func INT not null,
    cargo VARCHAR(40) not null,
    turno VARCHAR(20) not null,
    PRIMARY KEY (cpf),
    UNIQUE (id_func),
    FOREIGN KEY (cpf) REFERENCES pessoa
)

CREATE INDEX idx_cliente_cpf
```

```
ON cliente (cpf);

CREATE INDEX idx_funcionario_cpf
ON funcionario (cpf);

CREATE TABLE produtos (
    id INT not null,
    valor_venda MONEY not null,
    quant_estoque INT,
    tipo TINYINT NOT NULL,
    PRIMARY KEY (id)
)

CREATE TABLE fornecedor (
    cnpj VARCHAR(14) not null,
    nome VARCHAR(255) not null,
    email VARCHAR(255) not null,
    endereco VARCHAR(255) not null,
    tipo INT DEFAULT 1,
    PRIMARY KEY (cnpj)
)

CREATE TABLE editora (
    cnpj VARCHAR(14),
    PRIMARY KEY (cnpj),
    FOREIGN KEY (cnpj) REFERENCES fornecedor
)

CREATE INDEX idx_editora_cnpj
ON editora (CNPJ);

CREATE TABLE livros (
    id INT not null,
    isbn VARCHAR(13) not null,
    nome VARCHAR(50) not null,
    autor VARCHAR(50) not null,
    genero VARCHAR(50),
    cnpj VARCHAR(14) not null,
    PRIMARY KEY (id),
    UNIQUE (isbn),
    FOREIGN KEY (id) REFERENCES produtos,
    FOREIGN KEY (cnpj) REFERENCES editora
```

```
)

CREATE INDEX idx_livros_id
ON livros (id);

CREATE INDEX idx_livros_cnpj
ON livros (cnpj);

CREATE TABLE materiaisEscolares (
    id INT not null,
    modelo VARCHAR(50),
    marca VARCHAR(50) not null,
    descricao VARCHAR(100),
    PRIMARY KEY (id),
    FOREIGN KEY (id) REFERENCES produtos
)

CREATE INDEX idx_materiais_escolares_id
ON materiaisEscolares (id);

CREATE TABLE atendimento ( --ATUALIZADO
    codigo INT not null,
    cpf_cli VARCHAR(11) not null,
     cpf_func VARCHAR(11) not null,
    valor_total MONEY not null,
    data DATE,
    PRIMARY KEY (codigo),
    FOREIGN key (cpf_cli) REFERENCES cliente,
    FOREIGN key (cpf_func) REFERENCES funcionario
)

CREATE INDEX idx_atendimento_cpf_cli
ON atendimento (cpf_cli);

CREATE INDEX idx_atendimento_cpf_func
ON atendimento (cpf_func);

CREATE TABLE vender (
    codigo_atend INT not null,
    id_prod INT not null,
    quant_produtos INT not null,
    PRIMARY KEY (codigo_atend, id_prod),
```

```
    FOREIGN KEY (codigo_atend) REFERENCES atendimento,
    FOREIGN KEY (id_prod) REFERENCES produtos
)

CREATE INDEX idx_vender_codigo_atend
ON vender (codigo_atend);

CREATE INDEX idx_vender_id_prod
ON vender (id_prod);

create table alugar (
     codigo_atend INT not null,
     id_livro INT not null,
     devolucao DATE,
     valor_aluguel MONEY not null,
     multa MONEY,
     PRIMARY KEY (codigo_atend, id_livro),
     FOREIGN KEY (id_livro) REFERENCES livros,
     FOREIGN KEY (codigo_atend) REFERENCES atendimento
)

CREATE INDEX idx_alugar_codigo_atend
ON alugar (codigo_atend);

CREATE INDEX idx_alugar_id_livro
ON alugar (id_livro);

CREATE table fornecer (
    id_prod INT not null,
    cnpj VARCHAR(14) not null,
    primary KEY (id_prod, cnpj),
    FOREIGN KEY (id_prod) REFERENCES produtos,
    FOREIGN KEY (cnpj) REFERENCES fornecedor
)

CREATE INDEX idx_fornecer_id_prod
ON fornecer (id_prod);

CREATE INDEX idx_fornecer_cnpj
ON fornecer (cnpj);
```

6.1 Manipulação da Base de Dados

Script para a criação dos procedimentos armazenados e gatilhos para a realização das operações de manipulação da base de dados. Operações mínimas a serem implementadas: cadastrar cliente, cadastrar funcionário, cadastrar fornecedores, cadastrar livros, cadastrar material escolar, cadastrar atendimento, cadastrar aluguel e cadastrar devolução de livros.

VISÕES

– View LISTAR CLIENTES

```
CREATE VIEW viewClientes
AS
SELECT pessoa.cpf, pessoa.nome, pessoa.endereco, pessoa.email,
pessoa.telefone
FROM pessoa  INNER JOIN cliente
ON pessoa.cpf = cliente.cpf;
```

– View LISTAR FUNCIONARIOS

```
CREATE VIEW viewFuncionarios
AS
SELECT pessoa.cpf, pessoa.nome, pessoa.endereco, pessoa.email,
pessoa.telefone, funcionario.id_func, funcionario.cargo, funcionario.turno
FROM pessoa INNER JOIN funcionario
ON pessoa.cpf = funcionario.cpf;
```

– View LISTAR FORNECEDORES

```
CREATE VIEW viewFornecedores
AS
SELECT fornecedor.cnpj, fornecedor.nome, fornecedor.email,
fornecedor.endereco, fornecedor.tipo
FROM fornecedor INNER JOIN editora
ON fornecedor.cnpj = editora.cnpj;
```

– View MATERIAIS ESCOLARES

```
CREATE VIEW viewMateriais
AS
SELECT prod.id, prod.valor_venda, prod.quant_estoque, prod.tipo,
materiais.modelo, materiais.marca, materiais.descricao
FROM produtos prod INNER JOIN materiaisEscolares materiais
```

```
ON prod.id = materiais.id;
```

– View EDITORAS

```
CREATE VIEW viewEditoras
AS
SELECT fornecedor.cnpj, fornecedor.nome, fornecedor.email,
fornecedor.endereco, fornecedor.tipo
FROM fornecedor INNER JOIN editora
ON fornecedor.cnpj = editora.cnpj;
```

– View LIVROS

```
CREATE VIEW viewLivros
AS
SELECT prod.id, prod.valor_venda, prod.quant_estoque, prod.tipo, livros.isbn,
livros.nome, livros.autor, livros.genero, livros.cnpj
FROM produtos prod INNER JOIN livros
ON prod.id = livros.id;
```

PROCEDIMENTOS

CADASTRAR CLIENTE

```
CREATE PROCEDURE cadastrar_cliente
@cpf varchar (11),
@nome varchar(255),
@endereco varchar(255),
@email varchar(255),
@telefone varchar(15)
AS
BEGIN TRANSACTION
INSERT INTO pessoa VALUES (@cpf, @nome, @endereco, @email,
@telefone)
if @@rowcount > 0
     BEGIN
     INSERT INTO cliente (cpf)
     VALUES (@cpf)
     if @@rowcount > 0
          BEGIN
          COMMIT TRANSACTION
          END
     else
          BEGIN
```

```
          ROLLBACK TRANSACTION
          END
     END
else
     BEGIN
     ROLLBACK TRANSACTION
     END
```

CADASTRAR FUNCIONÁRIO

```
     CREATE PROCEDURE cadastrar_funcionario
     @cpf VARCHAR(11),
     @nome VARCHAR(50),
     @endereco VARCHAR(30),
     @email VARCHAR(30),
     @telefone VARCHAR(15),
     @id_func INT,
     @cargo VARCHAR(40),
     @turno VARCHAR(20)
     AS
     BEGIN TRANSACTION
     INSERT INTO pessoa
     VALUES (@cpf, @nome, @endereco, @email, @telefone)
          IF @@ROWCOUNT > 0
               BEGIN
               INSERT INTO funcionario
               VALUES (@cpf, @id_func, @cargo, @turno)
               IF @@ROWCOUNT > 0
                    BEGIN
                    COMMIT TRANSACTION
                    END
               ELSE
                    BEGIN
                    ROLLBACK TRANSACTION
                    END
               END
          ELSE
               BEGIN
               ROLLBACK TRANSACTION
               END
```

CADASTRAR FORNECEDORES

```
CREATE PROCEDURE cadastrar_fornecedor
@cnpj VARCHAR(14),
@nome VARCHAR(50),
@endereco VARCHAR(30),
@email VARCHAR(30),
@tipo tinyint
AS
BEGIN TRANSACTION
INSERT INTO fornecedor (cnpj, nome, endereco, email,
tipo)
VALUES (@cnpj, @nome, @endereco, @email, @tipo);
    IF @@ROWCOUNT > 0
     BEGIN
        IF @tipo = 1
          BEGIN
            INSERT INTO editora (cnpj)
            VALUES (@cnpj);
          END
     COMMIT TRANSACTION;
     END
    ELSE
     BEGIN
        ROLLBACK TRANSACTION;
     END
```

CADASTRAR LIVROS

```
CREATE PROCEDURE cadastrar_livros
@ISBN VARCHAR(13),
@nome VARCHAR(50),
@autor VARCHAR(50),
@genero VARCHAR(50),
@CNPJ_editora VARCHAR(14),
@quant_estoque INT,
@valor_venda MONEY
AS
BEGIN
    BEGIN TRANSACTION;
       INSERT INTO produtos (valor_venda, quant_estoque,
tipo)
    VALUES (@valor_venda, @quant_estoque, 1);
```

```
          IF @@ROWCOUNT > 0
          BEGIN
                INSERT INTO livros (isbn, nome, autor, genero,
cnpj)VALUES (@ISBN, @nome, @autor, @genero, @CNPJ_editora);
              IF @@ROWCOUNT > 0
              BEGIN
                  COMMIT TRANSACTION;
              END
              ELSE
              BEGIN
                  ROLLBACK TRANSACTION;
              END
          END
          ELSE
          BEGIN
              ROLLBACK TRANSACTION;
          END
      END;
```

CADASTRAR MATERIAL ESCOLAR

```
CREATE PROCEDURE cadastro_materialEscolar
    @id INT,
    @modelo VARCHAR(20),
    @marca VARCHAR(20),
    @tipo INT
AS
BEGIN
    BEGIN TRANSACTION;

    INSERT INTO materiaisEscolares (id, modelo, marca, tipo)
    VALUES (@id, @modelo, @marca, @tipo);

    IF @@ROWCOUNT > 0
    BEGIN
        COMMIT TRANSACTION;
    END
    ELSE
    BEGIN
        ROLLBACK TRANSACTION;
    END
END
```

CADASTRAR ATENDIMENTO

```
CREATE PROCEDURE cadastrar_atendimento
    @codigo INT,
    @CPF_cli VARCHAR(11),
    @CPF_func VARCHAR(11),
    @valor_total MONEY,
    @data DATE
AS
BEGIN
    BEGIN TRANSACTION;

        INSERT INTO atendimento (codigo, cpf_cli, cpf_func,
valor_total, data)
        VALUES (@codigo, @CPF_cli, @CPF_func, @valor_total,
@data);

    IF @@ROWCOUNT > 0
    BEGIN
        COMMIT TRANSACTION;
    END
    ELSE
    BEGIN
        ROLLBACK TRANSACTION;
    END;
END;
```

CADASTRAR ALUGUEL

```
CREATE PROCEDURE cadastrar_aluguel
    @devolucao DATE,
    @valor_aluguel MONEY,
    @multa MONEY,
    @nome_livro VARCHAR(255)
AS
BEGIN
    BEGIN TRANSACTION;
```

```
    DECLARE @quant_estoque INT;
       IF NOT EXISTS (SELECT 1 FROM livros WHERE nome =
@nome_livro)
    BEGIN
        PRINT 'Livro não encontrado';
        ROLLBACK TRANSACTION;
        RETURN;
    END

    SELECT @quant_estoque = quant_estoque
    FROM livros
    WHERE nome = @nome_livro;
    IF @quant_estoque > 0
    BEGIN
        INSERT INTO alugar (devolucao, valor_aluguel, multa)
        VALUES (@devolucao, @valor_aluguel, @multa);
        IF @@ROWCOUNT > 0
        BEGIN
            UPDATE livros
            SET quant_estoque = @quant_estoque - 1
            WHERE nome = @nome_livro;
            IF @@ROWCOUNT > 0
            BEGIN
                COMMIT TRANSACTION
            END
        END
    END
    ROLLBACK TRANSACTION;
END;
```

CADASTRAR DEVOLUÇÃO DO LIVRO

```
CREATE PROCEDURE cadastrar_devolucao
    @devolucao DATE,
    @valor_aluguel MONEY,
    @multa MONEY,
    @nome_livro VARCHAR(255)
AS
BEGIN
    BEGIN TRANSACTION;
```

```
    DECLARE @quant_estoque INT, @DataAtual DATE;

    SELECT @quant_estoque = quant_estoque
    FROM livros
    WHERE nome = @nome_livro;

    UPDATE livros
    SET quant_estoque = @quant_estoque + 1
    WHERE nome = @nome_livro;
    IF @@ROWCOUNT > 0
    BEGIN
        SELECT @DataAtual = GETDATE();
        IF @DataAtual > @devolucao
        BEGIN
              SET @multa = @multa + (DATEDIFF(DAY, @devolucao,
@DataAtual) * 4.99);
            INSERT INTO alugar
            VALUES (@devolucao, @valor_aluguel, @multa);
            IF @@ROWCOUNT > 0
            BEGIN
                COMMIT TRANSACTION;
            END
            ELSE
            BEGIN
                ROLLBACK TRANSACTION;
            END
        END
        ELSE
        BEGIN
            COMMIT TRANSACTION;
        END
    END
    ELSE
    BEGIN
        ROLLBACK TRANSACTION;
    END
END;
```

excluir material

```
CREATE PROCEDURE excluir_material
    @id_material INT
AS
```

```
BEGIN
    DELETE FROM materiaisEscolares
    WHERE id = @id_material;
END;
```

Excluir Livros

```
CREATE PROCEDURE excluir_livro
    @id_livro INT
AS
BEGIN
    DELETE FROM livros
    WHERE id = @id_livro;
END;
```

Atualizar Fornecedor

Unset

```
CREATE PROCEDURE atualizar_fornecedor
@cnpj VARCHAR(14),
@nome VARCHAR(255),
@email VARCHAR(255),
@endereco VARCHAR(255),
@tipo INT
AS
BEGIN
  UPDATE fornecedor
     SET nome = @nome, email = @email, endereco =
@endereco, tipo = @tipo
  WHERE cnpj = @cnpj;
END;
```

Atualizar Livros

```
CREATE PROCEDURE atualizar_livro
@id INT,
@nome VARCHAR(50),
@autor VARCHAR(50),
@genero VARCHAR(50)
AS
BEGIN
    UPDATE livros
    SET nome = @nome, autor = @autor, genero = @genero
    WHERE id = @id;
END;
```

Atualizar Produtos

Atual

GATILHOS

CADASTRAR VENDA

```
CREATE TRIGGER tr_CadastrarVenda
ON vender
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @codigo_atend INT, @id_prod INT, @quant_produtos
INT;

    SELECT @codigo_atend = i.codigo_atend,
           @id_prod = i.id_prod,
           @quant_produtos = i.quant_produtos
    FROM inserted i
```

```
      WHERE EXISTS (SELECT 1 FROM produtos p WHERE p.id =
i.id_prod)
      AND i.quant_produtos > 0;

    IF @@ROWCOUNT = 0
    BEGIN
        PRINT 'Produto não encontrado. Venda não processada.';
        RETURN;  -- Sair do trigger se o produto não existir
    END

    UPDATE produtos
    SET quant_estoque = quant_estoque - @quant_produtos
    WHERE id = @id_prod;


END;
```

DELETAR VENDA

```
CREATE TRIGGER tr_DeletarVenda
ON vender
AFTER DELETE
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @codigo_atend INT;

    SELECT @codigo_atend = deleted.codigo_atend
    FROM deleted;


    DELETE FROM atendimento
    WHERE codigo = @codigo_atend;


END;
```

ALTERAR VENDA

```
CREATE TRIGGER tr_AlterarVenda
ON vender
```

```
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;

      DECLARE @codigo_atend INT, @quant_produtos_antigo INT,
@quant_produtos_novo INT;

    SELECT @codigo_atend = i.codigo_atend,
           @quant_produtos_antigo = d.quant_produtos,
           @quant_produtos_novo = i.quant_produtos
    FROM inserted i
    INNER JOIN deleted d ON i.codigo_atend = d.codigo_atend;

      -- Atualizar o valor total na tabela atendimento se a
quantidade de produtos vendidos for alterada
    IF @quant_produtos_antigo <> @quant_produtos_novo
    BEGIN
        UPDATE atendimento
                        SET valor_total = valor_total -
(@quant_produtos_antigo * (SELECT valor_venda FROM produtos
WHERE id = i.id_prod))
                              + (@quant_produtos_novo * (SELECT
valor_venda FROM produtos WHERE id = i.id_prod))
        WHERE codigo = @codigo_atend;
    END

    END;
```