Universidade Estadual de Campinas - FT
Data 20/11/2023

Sistema de Gerenciamento de Locadora de Filmes

Disciplina: ST767 – Banco de Dados II
2º semestre de 2023
Profa. Gisele Busichia Baioco

Alunos:
Euler Breno de Oliveira, RA: 260783
Gino Carlo Graciano Grippo, RA: 248301
Thomas Landry De Almeida Silva. RA: 206302
João Gabriel Gomes Mariano, RA: 247487
Gabriel Sales Da Silva Matos De Araujo, RA: 248008

1. Descrição do sistema

Objetivo Geral

O banco de dados a ser desenvolvido é para um Sistema de Gerenciamento de Locadora de Filmes, projetado para controlar e otimizar as operações relacionadas ao aluguel de filmes. O objetivo principal deste sistema é oferecer uma plataforma eficiente para gerenciar o acervo de filmes, os clientes da locadora, as transações de aluguel e devolução, bem como proporcionar uma experiência de aluguel simplificada para os usuários. Além da locação, esse semestre o estabelecimento também começará a vender filmes.

Requisitos de dados

- **Catálogo de Filmes:** O sistema permitirá o cadastro e a manutenção de informações detalhadas sobre os filmes disponíveis na locadora

- **Pesquisa de elenco:** Na entrada da locadora, o consumidor pode pesquisar pelo integrante do filme, e todos os filmes em que ele aparece irão aparecer na tela, dizendo também os status de cada filme (se está disponível para locação, qual o preço do valor de locação diário e se está disponível para compra).

- **Cadastro de Clientes:** Será possível cadastrar novos clientes na locadora, coletando suas informações pessoais

- **Gerenciamento de funcionários**: O sistema deve permitir o cadastro de novos funcionários, dizendo em qual filial ele trabalha

- **Locações ministradas de forma alternativa:** Ao contrário de locadoras comuns, nesta locadora você pode alugar o filme por quantos dias quiser.

- **Gestão de Locações:** O sistema será capaz de registrar as locações de filmes para os clientes, incluindo a data de retirada, a data de devolução prevista e as penalidades por atraso.

- **Devolução de Filmes:** Os clientes poderão devolver os filmes alugados, e o sistema registra a data da devolução, calculando automaticamente as taxas de atraso, se aplicáveis.

- **Compra:** O sistema permite que os clientes comprem filmes

- **Filiais:** esta loja possui diversas lojas ao longo do país, portanto precisa saber quais são os funcionários de cada filial.

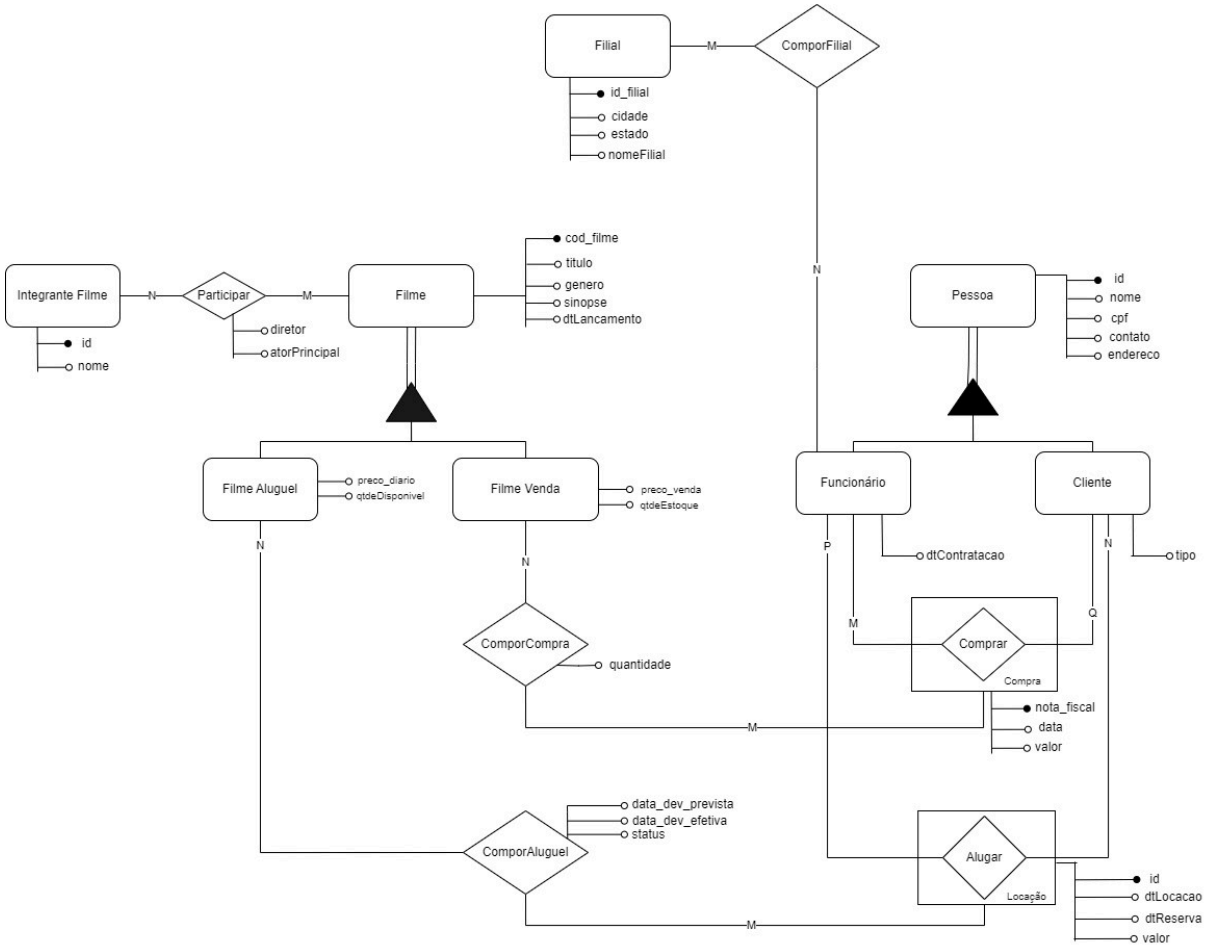
Lógica do sistema

- **Sobre a lógica da locação:**
  - Uma locação contém vários filmes, os quais tem seu preço determinado acordo com os seguintes fatores
  - Valor base do filme de aluguel = Preço diário (determinado pelo filme) multiplicado pela quantidade de Dias Emprestados
  - Se o filme não for entregue na data prevista, será adicionado ao valor da locação o cálculo do Preço diário multiplicado pela quantidade de Dias de atraso.
  - Se uma reserva for desejada, o valor de preço diário multiplicado pela quantidade de dias reservados será adicionado ao valor da locação.
  - Um filme poderá ser devolvido antes da data prevista para devolução, porém a quantidade de dias emprestados será determinada a partir da data de devolução prevista, e não da real.
  - Quando a locação possui o status "Ativo", ela reduzirá a quantidade de filmes disponíveis.
  - Quando a locação possui o status "Reservado", ela reduzirá a quantidade de filmes disponíveis.
  - Quando a locação possui o status "Finalizado", ela aumentará a quantidade de filmes disponíveis.
  - Quando a locação possui o status "Cancelado", ela também aumentará a quantidade de filmes disponíveis.

- **Sobre a lógica da venda:**
  - Os estoques dos filmes são divididos em filmes destinados a venda e filmes destinados a locação, portanto são administrados independentemente
  - A compra de um ou mais filmes também possui um valor total, o qual irá variar de acordo com a quantidade de filmes e o preço deles
    - Valor = Quantidade * Preço de Venda
  - Ademais, quando um filme é comprado, a quantidade de estoque daquele filme será reduzida

- **Sobre o tipo de cliente**
  - O cliente tem a opção de pagar uma mensalidade para a locadora, a qual determinará se o cliente é do tipo Comum ou Premium. Se for Premium, ele terá a oportunidade de receber descontos ao fazer compras ou locações.

2. Projeto Conceitual do Banco de Dados



3. Projeto Lógico do Banco de Dados

- **Pessoa** = {<u>id</u>, nome, cpf, contato, endereco}

- **Funcionário** = {<u>id</u>, dtContratacao}
  - id → chave estrangeira referenciando Pessoa

- **Cliente** = {<u>id</u>, tipo}
  - <u>id</u>→ chave estrangeira referenciando Pessoa

- **Compra** = {<u>nota_fiscal</u>, data, valor, id_funcionario, id_cliente}
  - id_funcionario -> chave estrangeira referenciando Funcionario
  - id_cliente -> chave estrangeira rerenciando Cliente

- **Locação** = {<u>id</u>, data, dtLocacao, dtReserva, valor, id_funcionario , id_cliente}
  - id_funcionario -> chave estrangeira referenciando Funcionario

    - id_cliente -> chave estrangeira rerenciando Cliente

- **Filme** = {cod_filme, titulo, genero, sinopse, dt_lancamento}

- **Filme_venda** = {cod_filme, preco_venda, qtdeEstoque}
    - cod_filme → chave estrangeira referenciando Filme

- **ComporCompra** = {cod_filme, nota_fiscal, quantidade}
    - cod_filme → chave estrangeira referenciando Filme Venda
    - nota_fiscal → chave estrangeira referenciando Compra

- **Filme_aluguel** = {cod_filme, preco_diario,qtdDisponivel}
    - cod_filme → chave estrangeira referenciando Filme

- **ComporAluguel** = {cod_filme, id, data_dev_prevista, data_dev_efetiva, status}
    - cod_filme → chave estrangeira referenciando Filme Aluguel
    - id → chave estrangeira referenciando Locação

- **IntegranteFilme** = {id, nome}

- **Participação** = {cod_filme, id, diretor, atorPrincipal}
    - cod_filme = chave estrangeira referenciando Filme
    - id = chave estrangeira referenciando IntegranteFilme

- **Filial** = {id_filial, cidade, estado, nomeFilial}

- **ComporFilial** = {id_filial, id}
    - id_filial = chave estrangeira referenciando Filial
    - id = chave estrangeira referenciando Funcionario

4. Projeto Físico do Banco de Dados

```
Unset
CREATE TABLE Pessoa (
id INT NOT NULL,
nome VARCHAR(255) NOT NULL,
cpf VARCHAR(11) NOT NULL,
contato VARCHAR(255) NOT NULL,
endereco VARCHAR(255) NOT NULL,
PRIMARY KEY (id)
)
go

CREATE TABLE Funcionario (
id INT NOT NULL,
dtContratacao DATE NOT NULL,
```

```
PRIMARY KEY (id),
FOREIGN KEY (id) REFERENCES Pessoa
)
go

CREATE TABLE Cliente (
id INT NOT NULL,
tipo VARCHAR(255) NOT NULL,
PRIMARY KEY (id),
FOREIGN KEY (id) REFERENCES Pessoa
)
go

CREATE TABLE Compra (
nota_fiscal VARCHAR(255) NOT NULL,
data DATE NOT NULL,
valor DECIMAL(10,2) NOT NULL,
id_funcionario INT NOT NULL,
id_cliente INT NOT NULL,
PRIMARY KEY (nota_fiscal),
FOREIGN KEY (id_funcionario) REFERENCES Funcionario (id),
FOREIGN KEY (id_cliente) REFERENCES Cliente (id)
)
go

create index ixcompra_cli on Compra(id_cliente)
go
create index ixcomra_func on Compra(id_funcionario)
go


CREATE TABLE Locacao (
id INT NOT NULL,
dtLocacao DATE,
dtReserva DATE,
id_funcionario INT NOT NULL,
id_cliente INT NOT NULL,
valor DECIMAL(10,2) NOT NULL
PRIMARY KEY (id),
FOREIGN KEY (id_funcionario) REFERENCES Funcionario (id),
FOREIGN KEY (id_cliente) REFERENCES Cliente (id)
)
go

create index ixlocacao_cli on Locacao(id_cliente)
go
create index ixlocacao_func on Locacao(id_funcionario)
go
```

```
CREATE TABLE Filme (
cod_filme INT NOT NULL,
titulo VARCHAR(255) NOT NULL,
genero VARCHAR(255) NOT NULL,
sinopse VARCHAR(255) NOT NULL,
dt_lancamento DATE NOT NULL,
PRIMARY KEY (cod_filme)
)
go

CREATE TABLE Filme_venda (
cod_filme INT NOT NULL,
preco_venda DECIMAL(10,2) NOT NULL,
qtdeEstoque INT NOT NULL,
PRIMARY KEY (cod_filme),
FOREIGN KEY (cod_filme) REFERENCES Filme (cod_filme)
)
go

CREATE TABLE ComporCompra (
cod_filme INT NOT NULL,
nota_fiscal VARCHAR(255) NOT NULL,
quantidade INT NOT NULL,
PRIMARY KEY (cod_filme, nota_fiscal),
FOREIGN KEY (cod_filme) REFERENCES Filme_venda (cod_filme),
FOREIGN KEY (nota_fiscal) REFERENCES Compra (nota_fiscal)
)
go

create index ixcompra_filme on ComporCompra(cod_filme)
go

create index ixcompra_nota on ComporCompra(nota_fiscal)
go

CREATE TABLE Filme_aluguel (
cod_filme INT NOT NULL,
preco_diario DECIMAL(10,2) NOT NULL,
qtdDisponivel INT NOT NULL,
PRIMARY KEY (cod_filme),
FOREIGN KEY (cod_filme) REFERENCES Filme (cod_filme)
)
go

CREATE TABLE ComporAluguel (
cod_filme INT NOT NULL,
idLocacao INT NOT NULL,
status varchar(15) NOT NULL,
data_dev_prevista DATE NOT NULL,
```

```
data_dev_efetiva DATE,
PRIMARY KEY (cod_filme, idLocacao),
FOREIGN KEY (cod_filme) REFERENCES Filme_aluguel (cod_filme),
FOREIGN KEY (idLocacao) REFERENCES Locacao (id)
)
go

create index ixaluguel_filme on ComporAluguel(cod_filme)
go

create index ixaluguel_locaca on ComporAluguel(idLocacao)
go


CREATE TABLE IntegranteFilme (
id INT NOT NULL,
nome VARCHAR(255) NOT NULL,
PRIMARY KEY (id)
)
go

CREATE TABLE Participacao (
cod_filme INT NOT NULL,
id INT NOT NULL,
diretor varchar(3) NOT NULL,
atorPrincipal varchar(3) NOT NULL,
PRIMARY KEY (cod_filme, id),
FOREIGN KEY (cod_filme) REFERENCES Filme (cod_filme),
FOREIGN KEY (id) REFERENCES IntegranteFilme (id)
)
go

create index ixparticipacao_integrante on Participacao(id)
go

create index ixparticipacao_filme on Participacao(cod_filme)
go

CREATE TABLE Filial (
id_filial INT NOT NULL,
cidade VARCHAR(30) NOT NULL,
estado VARCHAR(30) NOT NULL,
nomeFilial VARCHAR(255) NOT NULL
PRIMARY KEY (id_filial)
)
go

CREATE TABLE ComporFilial (
id_filial INT NOT NULL,
```

```
id INT NOT NULL
PRIMARY KEY (id_filial, id),
FOREIGN KEY (id_filial) REFERENCES Filial (id_filial),
FOREIGN KEY (id) REFERENCES Funcionario (id)
)
go

create index ixcompor_filial on ComporFilial(id_filial)
go

create index ixfilial_funcionario on ComporFilial(id)
go
```

5. Manipulação do Banco de Dados:

- **Views**
  - Relatório de vendas geral: quantos filmes foram vendidos no mês (organizado por dia) e no ano (organizado por mês), sem detalhes dos filmes, apenas a quantidade geral

```
Unset
CREATE VIEW RelatorioVendasGeral
AS
SELECT COUNT(C.nota_fiscal) AS 'Quantidade de Vendas', DAY(C.data) AS Dia,
MONTH(C.data) AS Mês, YEAR(C.data) AS Ano
FROM Compra C
GROUP BY DAY(C.data), MONTH(C.data), YEAR(C.data)
go
```

  - Relatório de locações geral: semelhante ao de vendas, apresenta quantos filmes foram alugados no mês (organizado por dia) e no ano (organizado por mês), sem detalhes dos filmes, apenas a quantidade geral

```
Unset
CREATE VIEW RelatorioLocacoesGeral
AS
SELECT COUNT(L.id) AS 'Quantidade de Alugueis', DAY(L.data) AS Dia, MONTH(L.data)
AS Mês, YEAR(L.data) AS Ano
FROM Locacao L
GROUP BY DAY(L.data), MONTH(L.data), YEAR(L.data)
go
```

- Relatório de filmes: titulo do filme, quantidade de locações e quantidade de vendas

```
Unset
CREATE VIEW RelatorioFilmes AS
SELECT
  F.cod_filme AS 'Codigo Filme',
  F.titulo AS 'Titulo',
  COUNT(DISTINCT CA.idLocacao) AS 'Quantidade Locacoes',
  COUNT(DISTINCT CC.nota_fiscal) AS 'Quantidade Vendas'
FROM
  Filme F
LEFT JOIN
  ComporAluguel CA ON F.cod_filme = CA.cod_filme
LEFT JOIN
  ComporCompra CC ON F.cod_filme = CC.cod_filme
GROUP BY
  F.cod_filme, F.titulo
go
```

- Relatório de gerenciamento de funcionários: deve mostrar o ID do funcionário, seu nome, e quantos aluguéis e vendas ele promoveu

```
Unset
CREATE VIEW RelatorioGerenciamentoFuncionarios AS
SELECT
  F.id AS 'ID Funcionario',
  P.nome AS 'Nome',
  COUNT(DISTINCT L.id) AS 'Locacoes',
  COUNT(DISTINCT C.nota_fiscal) AS 'Vendas'
FROM
  Funcionario F
INNER JOIN
  Pessoa P ON P.id = F.id
INNER JOIN
  Compra C ON F.id = C.id_funcionario
INNER JOIN
  Locacao L ON F.id = L.id_funcionario
GROUP BY
  F.id, P.nome
```

```
go
```

- Relatório de gerenciamento de filiais: deve mostrar o nome da filial, a quantidade de funcionários, a quantidade de vendas e a quantidade de locações

```
Unset
CREATE VIEW RelatorioGerenciamentoFiliais AS
SELECT
  F.id_filial AS 'ID Filial',
  Filial.nomeFilial AS 'Nome Filial',
  COUNT(DISTINCT CF.id) AS 'QuantidadeFuncionarios',
  COUNT(DISTINCT C.nota_fiscal) AS 'QuantidadeVendas',
  COUNT(DISTINCT L.id) AS 'QuantidadeLocacoes'
FROM
  Filial
LEFT JOIN
  ComporFilial CF ON Filial.id_filial = CF.id_filial
LEFT JOIN
  Funcionario F ON CF.id = F.id
LEFT JOIN
  Compra C ON F.id = C.id_funcionario
LEFT JOIN
  Locacao L ON F.id = L.id_funcionario
GROUP BY
  F.id_filial, Filial.nomeFilial
go
```

- Relatório de estados: deve mostrar a quantidade de vendas e a quantidade de locações por estado

```
Unset
CREATE VIEW RelatorioEstados AS
SELECT
  Filial.estado AS 'Estado',
  COUNT(DISTINCT C.nota_fiscal) AS 'QuantidadeVendas',
  COUNT(DISTINCT L.id) AS 'QuantidadeLocacoes'
FROM
  Filial
LEFT JOIN
  ComporFilial CF ON Filial.id_filial = CF.id_filial
LEFT JOIN
```

```
  Funcionario F ON CF.id = F.id
LEFT JOIN
  Compra C ON F.id = C.id_funcionario
LEFT JOIN
  Locacao L ON F.id = L.id_funcionario
GROUP BY
  Filial.estado
go
```

- Relatório de cidades: deve mostrar a quantidade de vendas e a quantidade de locações por cidade e por estado

```
Unset
CREATE VIEW RelatorioCidades AS
SELECT
  Filial.cidade AS 'Cidade',
  Filial.estado AS 'Estado',
  COUNT(DISTINCT C.nota_fiscal) AS 'QuantidadeVendas',
  COUNT(DISTINCT L.id) AS 'QuantidadeLocacoes'
FROM
  Filial
LEFT JOIN
  ComporFilial CF ON Filial.id_filial = CF.id_filial
LEFT JOIN
  Funcionario F ON CF.id = F.id
LEFT JOIN
  Compra C ON F.id = C.id_funcionario
LEFT JOIN
  Locacao L ON F.id = L.id_funcionario
GROUP BY
  Filial.cidade, Filial.estado
go
```

- Relatório de estoque de filmes: deve mostrar a quantidade disponível de filmes para aluguel e filmes para compra em estoque, além de seus respectivos títulos e códigos.

```
Unset
CREATE VIEW RelatorioEstoqueFilmes AS
SELECT F.cod_filme as 'Codigo Filme', F.titulo as 'Titulo', FV.qtdeEstoque as
'Disponivel para venda', FA.qtdDisponivel as 'Disponivel para aluguel'
```

```
FROM Filme F
INNER JOIN Filme_venda FV ON F.cod_filme = FV.cod_filme
INNER JOIN Filme_aluguel FA ON F.cod_filme = FA.cod_filme
go
```

- Pesquisa Local: Na entrada da locadora, o consumidor pode pesquisar pelo integrante do filme, e todos os filmes em que ele aparece irão aparecer na tela, dizendo também os status de cada filme (se está disponível para locação, qual o preço do valor de locação diário e se está disponível para compra).

```
Unset
CREATE VIEW PesquisaLocal AS
SELECT DISTINCT
  F.cod_filme AS 'Código do Filme',
  F.titulo AS 'Título',
  F.genero AS 'Gênero',
  F.sinopse AS 'Sinopse',
  F.dt_lancamento AS 'Data de Lançamento',
  FA.preco_diario AS 'Preço Locação Diário',
  FA.qtdDisponivel AS 'Quantidade Disponível Locação',
  FV.preco_venda AS 'Preço Venda',
  FV.qtdeEstoque AS 'Quantidade Disponível Venda'
FROM
  Filme F
LEFT JOIN
  Filme_aluguel FA ON F.cod_filme = FA.cod_filme
LEFT JOIN
  Filme_venda FV ON F.cod_filme = FV.cod_filme
LEFT JOIN
  ComporAluguel CA ON F.cod_filme = CA.cod_filme
LEFT JOIN
  ComporCompra CC ON F.cod_filme = CC.cod_filme
LEFT JOIN
  Locacao L ON CA.idLocacao = L.id
LEFT JOIN
  Compra C ON CC.nota_fiscal = C.nota_fiscal
LEFT JOIN
  Participacao P ON F.cod_filme = P.cod_filme
LEFT JOIN
  IntegranteFilme I ON P.id = I.id
go
```

- **Procedimentos Armazenados ((Stored Procedures): para implementação das transações relacionadas às funcionalidades de inclusão, alteração e exclusão de dados;**
    - InserirLocacao

```
Unset
CREATE PROCEDURE InserirLocacao
	@idLocacao INT,
	@dtLocacao DATE,
	@idFuncionario INT,
	@idCliente INT,
	@valor NUMERIC(5,2)
AS
BEGIN
	BEGIN TRANSACTION

		INSERT INTO Locacao (id, dtLocacao, id_funcionario, id_cliente,
valor)
		VALUES (@idLocacao, @dtLocacao, @idFuncionario, @idCliente,
@valor)

		IF @@ROWCOUNT > 0
		BEGIN
			COMMIT TRANSACTION
			RETURN 1
		END
		ELSE
		BEGIN
			ROLLBACK TRANSACTION
			RETURN 0
		END
END
```

    - InserirReserva

```
Unset
CREATE PROCEDURE InserirReserva
	@idReserva INT,
	@dtLocacao DATE,
	@idFuncionario INT,
	@idCliente INT,
	@valor NUMERIC(5,2)
AS
BEGIN
	BEGIN TRANSACTION
```

```
            INSERT INTO Locacao (id, dtReserva, id_funcionario, id_cliente,
valor)
            VALUES (@idReserva, @dtLocacao, @idFuncionario, @idCliente,
@valor)

            IF @@ROWCOUNT > 0
            BEGIN
                  COMMIT TRANSACTION
                  RETURN 1
            END
            ELSE
            BEGIN
                  ROLLBACK TRANSACTION
                  RETURN 0
            END
END
```

- InserirCliente

```
Unset
CREATE PROCEDURE InserirCliente
  @id int,
      @nome varchar(255),
      @cpf varchar(11),
      @contato varchar(255),
      @endereco varchar(255),
      @tipo VARCHAR(255)
AS
BEGIN TRANSACTION

  IF NOT EXISTS (SELECT 1 FROM Pessoa WHERE Pessoa.id = @id)
  BEGIN
    INSERT INTO Pessoa
    VALUES (@id, @nome, @nome, @cpf, @contato, @endereco)
  END

  INSERT INTO Cliente
  VALUES (@id, @tipo)

      IF @@ROWCOUNT > 1
      BEGIN
            COMMIT TRANSACTION
            RETURN 1
      END
```

```
        ELSE
        BEGIN
                ROLLBACK TRANSACTION
                RETURN 0
        END
go
```

- InserirFuncionario

```
Unset
CREATE PROCEDURE InserirFuncionario
  @id int,
        @nome varchar(255),
        @cpf varchar(11),
        @contato varchar(255),
        @endereco varchar(255),
        @dtContratacao DATE
AS
BEGIN TRANSACTION

  IF NOT EXISTS (SELECT 1 FROM Pessoa WHERE Pessoa.id = @id)
  BEGIN
    INSERT INTO Pessoa
    VALUES (@id, @nome, @nome, @cpf, @contato, @endereco)
  END

  INSERT INTO Funcionario
  VALUES (@id, @dtContratacao)

        IF @@ROWCOUNT > 1
        BEGIN
                COMMIT TRANSACTION
                RETURN 1
        END
        ELSE
        BEGIN
                ROLLBACK TRANSACTION
                RETURN 0
        END
go
```

- InserirFilmeAluguel

```
Unset

CREATE PROCEDURE InserirFilmeAluguel
  @cod_filme INT,
  @titulo VARCHAR(255),
  @genero VARCHAR(255),
  @sinopse VARCHAR(255),
  @dt_lancamento DATE,
  @preco_diario DECIMAL(10,2),
  @qtdDisponivel INT
AS
BEGIN TRANSACTION

  IF NOT EXISTS (SELECT 1 FROM Filme WHERE cod_filme = @cod_filme)
  BEGIN
    INSERT INTO Filme (cod_filme, titulo, genero, sinopse, dt_lancamento)
    VALUES (@cod_filme, @titulo, @genero, @sinopse, @dt_lancamento)
  END

  INSERT INTO Filme_aluguel (cod_filme, preco_diario, qtdDisponivel)
  VALUES (@cod_filme, @preco_diario, @qtdDisponivel)

        IF @@ROWCOUNT > 1
        BEGIN
               COMMIT TRANSACTION
               RETURN 1
        END
        ELSE
        BEGIN
               ROLLBACK TRANSACTION
               RETURN 0
        END
go
```

- InserirFilmeVenda

```
Unset
CREATE PROCEDURE InserirFilmeVenda
  @cod_filme INT,
  @titulo VARCHAR(255),
  @genero VARCHAR(255),
  @sinopse VARCHAR(255),
  @dt_lancamento DATE,
  @preco_venda DECIMAL(10,2),
  @qtdeEstoque INT
AS
```

```
BEGIN TRANSACTION

  IF NOT EXISTS (SELECT 1 FROM Filme WHERE cod_filme = @cod_filme)
  BEGIN
    INSERT INTO Filme (cod_filme, titulo, genero, sinopse, dt_lancamento)
    VALUES (@cod_filme, @titulo, @genero, @sinopse, @dt_lancamento)
  END

  INSERT INTO Filme_venda (cod_filme, preco_venda, qtdeEstoque)
  VALUES (@cod_filme, @preco_venda, @qtdeEstoque)

  IF @@ROWCOUNT > 0
  BEGIN
    COMMIT TRANSACTION
    RETURN 1
  END
  ELSE
  BEGIN
    ROLLBACK TRANSACTION
    RETURN 0
  END

go
```

- EntregarFilme

```
Unset
CREATE PROCEDURE EntregarFilme
	@idLocacao int,
	@codFilme int,
	@dtDevefetiva date
	AS
	BEGIN TRANSACTION
		UPDATE ComporAluguel SET status = 'Finalizado', data_dev_efetiva =
@dtDevefetiva WHERE idLocacao = @idLocacao and cod_filme = @codFilme
		if @@ROWCOUNT > 0
		BEGIN
			COMMIT TRANSACTION
			return 1
		END
		else
		BEGIN
			ROLLBACK TRANSACTION
			return 0
		END
```

- CancelarFilme

```
Unset
CREATE PROCEDURE CancelarFilme
	@idLocacao int,
	@codFilme int,
	@dtDevefetiva date
	AS
	BEGIN TRANSACTION
		UPDATE ComporAluguel SET status = 'Cancelado', data_dev_efetiva =
@dtDevefetiva WHERE idLocacao = @idLocacao and cod_filme = @codFilme
		if @@ROWCOUNT > 0
		BEGIN
			COMMIT TRANSACTION
			return 1
		END
		else
		BEGIN
			ROLLBACK TRANSACTION
			return 0
		END
```

- InserirFilmeNaLocacao

```
Unset
CREATE PROCEDURE InserirFilmeNaLocacao
	@codFilme int,
	@idLocacao int,
	@dtDevPrevista date
	AS
	BEGIN TRANSACTION
		INSERT INTO ComporAluguel VALUES (@codFilme, @idLocacao, 'Ativo',
@dtDevPrevista)
		if @@ROWCOUNT > 0
		BEGIN
			COMMIT TRANSACTION
			return 1
		END
		else
		BEGIN
			ROLLBACK TRANSACTION
			return 0
		END
```

- InserirFilmeNaReserva

```
Unset
	CREATE PROCEDURE InserirFilmeNaReserva
		@codFilme int,
		@idLocacao int,
		@dtDevPrevista date
		AS
		BEGIN TRANSACTION
			INSERT INTO ComporAluguel VALUES (@codFilme, @idLocacao,
	'Reservado', @dtDevPrevista)
			if @@ROWCOUNT > 0
			BEGIN
				COMMIT TRANSACTION
				return 1
			END
			else
			BEGIN
				ROLLBACK TRANSACTION
				return 0
			END
```

- **Gatilhos (Triggers): devem ser usados quando se julgar necessários**
  - Quando a data de entrega efetiva do filme na locação for atualizada (filme foi entregue), o valor total deve ser calculado

```
Unset
CREATE TRIGGER tgr_calcValorTot
ON ComporAluguel
FOR UPDATE
AS
BEGIN
  BEGIN TRANSACTION
  DECLARE @fin VARCHAR(15)
  DECLARE @can VARCHAR(15)
  declare @status varchar(15)
  SELECT @status = status from inserted
  IF (@status = 'Finalizado')
  BEGIN
    UPDATE Locacao
           SET  valor  =  (DATEDIFF(day,  Locacao.dtLocacao,  Locacao.data)  *
Filme_aluguel.preco_diario)
    FROM Locacao
    JOIN Filme_aluguel ON Locacao.id = Filme_aluguel.cod_filme
    JOIN deleted ON Locacao.id = deleted.id
```

```
    IF (@@rowcount > 0)
      COMMIT TRANSACTION
    ELSE
      ROLLBACK TRANSACTION
  END
  IF (@status = 'Reservado')
  BEGIN
    DECLARE @clienteTipo VARCHAR(15)
    SELECT @clienteTipo = Cliente.tipo
    FROM Cliente
    WHERE Cliente.id = (SELECT id_cliente FROM Locacao WHERE id = deleted.id)

    IF (@clienteTipo = 'Premium')
    BEGIN
      UPDATE Locacao
                SET  valor  =  (DATEDIFF(day,  Locacao.dtReserva,  Locacao.data)  *
Filme_aluguel.preco_diario * 0.75)
      FROM Locacao
      JOIN Filme_aluguel ON Locacao.id = Filme_aluguel.cod_filme
      JOIN deleted ON Locacao.id = deleted.id
      IF (@@rowcount > 0)
        COMMIT TRANSACTION
      ELSE
        ROLLBACK TRANSACTION
    END
    ELSE
    BEGIN
      UPDATE Locacao
                SET  valor  =  (DATEDIFF(day,  Locacao.dtReserva,  Locacao.data)  *
Filme_aluguel.preco_diario)
      FROM Locacao
      JOIN Filme_aluguel ON Locacao.id = Filme_aluguel.cod_filme
      JOIN deleted ON Locacao.id = deleted.id
      IF (@@rowcount > 0)
        COMMIT TRANSACTION
      ELSE
        ROLLBACK TRANSACTION
    END
  END
  ELSE
    COMMIT TRANSACTION

END
```

- Se a compra for excluída, aumentar o estoque de filme de venda

```
Unset
CREATE TRIGGER tgr_aumentaEstoqueVenda
ON ComporCompra
FOR DELETE
AS
BEGIN
  BEGIN TRANSACTION
  UPDATE Filme_venda
  SET qtdeEstoque = qtdeEstoque + 1
  FROM Filme_venda
  JOIN deleted ON Filme_aluguel.cod_filme = deleted.cod_filme
  IF (@@rowcount > 0)
    COMMIT TRANSACTION
  ELSE
    ROLLBACK TRANSACTION
END
```

- Se o aluguel for excluído, aumentar o estoque de filme de aluguel

```
Unset
CREATE TRIGGER tgr_aumentaEstoqueAluguel
ON ComporAluguel
FOR DELETE
AS
BEGIN
  BEGIN TRANSACTION
  UPDATE Filme_aluguel
  SET qtdDisponivel = qtdDisponivel + 1
  FROM Filme_aluguel
  JOIN deleted ON Filme_aluguel.cod_filme = deleted.cod_filme
  IF (@@rowcount > 0)
    COMMIT TRANSACTION
  ELSE
    ROLLBACK TRANSACTION
END
```

- Se um filme for excluído, deletar todos os seus participantes

```
Unset
CREATE TRIGGER tgr_deletaParticipantes
ON Filme
```

```
FOR DELETE
AS
BEGIN
  BEGIN TRANSACTION
  DELETE FROM Participação
  WHERE Participação.cod_filme IN (SELECT cod_filme FROM deleted)

  IF (@@ROWCOUNT > 0)
    COMMIT TRANSACTION
  ELSE
    ROLLBACK TRANSACTION
END
```

- Se um integrante for excluído, deletar todas as suas participações

```
Unset
CREATE TRIGGER tgr_deletaParticipações
ON IntegranteFilme
FOR DELETE
AS
BEGIN
  BEGIN TRANSACTION
  DELETE FROM Participação
  WHERE Participação.id IN (SELECT id FROM deleted)

  IF (@@ROWCOUNT > 0)
    COMMIT TRANSACTION
  ELSE
    ROLLBACK TRANSACTION
END
```

- Aumentar ou diminuir quantidade de itens do estoque de aluguel dependendo de cada status de empréstimo;

```
Unset
CREATE TRIGGER tgr_aumentaEstoqueAluguelStatus
ON ComporAluguel
FOR UPDATE
AS
BEGIN
  BEGIN TRANSACTION
  DECLARE @fin VARCHAR(15)
  DECLARE @can VARCHAR(15)
```

```
  declare @status varchar(15)
  SELECT @status = status from inserted
  IF (@status = 'Finalizado' OR @status = 'Cancelado')
  BEGIN
    UPDATE Filme_aluguel
    SET qtdDisponivel = qtdDisponivel + 1
    FROM Filme_aluguel
    JOIN deleted ON Filme_aluguel.cod_filme = deleted.cod_filme
    IF (@@rowcount > 0)
      COMMIT TRANSACTION
    ELSE
      ROLLBACK TRANSACTION
  END
  ELSE
    COMMIT TRANSACTION
END
```

- Quando um cliente comprar ou alugar um item, diminuir a o valor no estoque;

```
Unset
CREATE TRIGGER tgr_diminuiEstoqueVenda
ON ComporCompra
AFTER INSERT
AS
BEGIN
  BEGIN TRANSACTION
  UPDATE Filme_venda
  SET qtdeEstoque = qtdeEstoque - 1
  FROM Filme_venda
  JOIN inserted  ON Filme_venda.cod_filme = deleted.cod_filme
  WHERE Filme_venda.cod_filme IN (SELECT cod_filme FROM inserted)

  IF (@@rowcount > 0)
    COMMIT TRANSACTION
  ELSE
    ROLLBACK TRANSACTION
END
CREATE TRIGGER tgr_diminuiEstoqueAluguel
ON ComporAluguel
AFTER INSERT
AS
BEGIN
  BEGIN TRANSACTION
  UPDATE Filme_aluguel
  SET qtdDisponivel = qtdDisponivel - 1
```

```
  FROM Filme_aluguel
  JOIN inserted ON Filme_aluguel.cod_filme = deleted.cod_filme
  WHERE Filme_aluguel.cod_filme IN (SELECT cod_filme FROM inserted)

  IF (@@rowcount > 0)
    COMMIT TRANSACTION
  ELSE
    ROLLBACK TRANSACTION
END
```

- Atualizar automaticamente o status de um empréstimo para "cancelado" após 5 dias e adicionar uma multa;

```
Unset
CREATE TRIGGER tgr_atualizaStatusCancelado
ON ComporAluguel
AFTER UPDATE
AS
BEGIN
  SET NOCOUNT ON;

  DECLARE @dataAtual DATE = GETDATE();
  DECLARE @idLocacao INT;
  DECLARE @dtLocacao DATE;
  DECLARE @data_dev_prevista DATE;
  DECLARE @statusAnterior VARCHAR(15);
  DECLARE @statusAtual VARCHAR(15);

  SELECT @idLocacao = idLocacao, @data_dev_prevista = data_dev_prevista,
@statusAnterior = inserted.status, @statusAtual = inserted.status
  FROM inserted;

  IF (@statusAnterior <> @statusAtual AND @statusAtual IN ('Finalizado',
'Cancelado') AND @data_dev_prevista IS NOT NULL)
  BEGIN
    DECLARE @diasAtraso INT = DATEDIFF(day, @data_dev_prevista, @dataAtual);

    IF (@diasAtraso > 5)
    BEGIN
      UPDATE ComporAluguel
      SET status = 'Cancelado'
      WHERE idLocacao = @idLocacao;
    END
  END
END
```