1. Objetivo geral

Projetar um banco de dados para um Cinema, em um sistema que permita a compra de ingressos, bebidas e alimentos; registro de informações das salas; informações sobre as sessões; avaliação do usuário sobre os filmes em cartaz; gerenciamento do estoque do estabelecimento e dos funcionários.

1.1 Requisitos de dados

Quanto às salas de exibição:

- Um cinema tem muitas salas;
- Cada sala deve possuir um nome único e pode ser do tipo normal, IMAX ou VIP;
- Cada sala possui uma capacidade definida, com assentos numerados;
- Pode exibir diversos filmes, contanto que não seja no mesmo horário;

Quanto aos filmes exibidos:

- Um filme é exibido em uma sala e em um determinado horário, chamado de sessão;
- O cinema somente possui sessões nos horários: 14:00, 14:30, 16:30, 18:00, 19:30, 20:00, 22:00 e 24:00.
- Possuem nome em português, nome original, diretor, ano de lançamento, gênero, classificação indicativa, sinopse, premiação, avaliação.

Quanto aos funcionários:

- Possui carteira de trabalho (que o identifica unicamente), nome, data de admissão e salário.
- Um atendente pode intermediar a compra de ingresso de vários clientes;
- Um atendente deve intermediar a compra de diversos alimentos e bebidas por parte do cliente;
- Vários técnicos de manutenção podem realizar a manutenção de várias salas;
- Vários auxiliares de limpeza podem realizar a limpeza de várias salas.

Quanto aos ingressos:

- Possui sala da sessão, assento escolhido, modalidade (meia ou inteira) e horário da sessão;
- Possui valor diferente de acordo com a sala da sessão (normal, IMAX ou VIP);
- Não é possível um único ingresso ser de uma sessão regular, IMAX e VIP ao mesmo tempo.

Quanto aos clientes:

- Pode comprar um ou vários ingressos para uma ou várias sessões;
- Pode comprar no local, intermediado por um atendente ou online;
- Pode comprar alimentos e bebidas somente presencialmente, intermediado por um atendente.
- Possui cadastro com CPF, nome, data de nascimento e email.

Quanto aos fornecedores:

- Possuem cadastrado CNPJ, razão social, CEP, telefone e email;
- Vários alimentos vendidos no cinema são abastecidos por vários fornecedores.

Quanto aos alimentos vendidos:

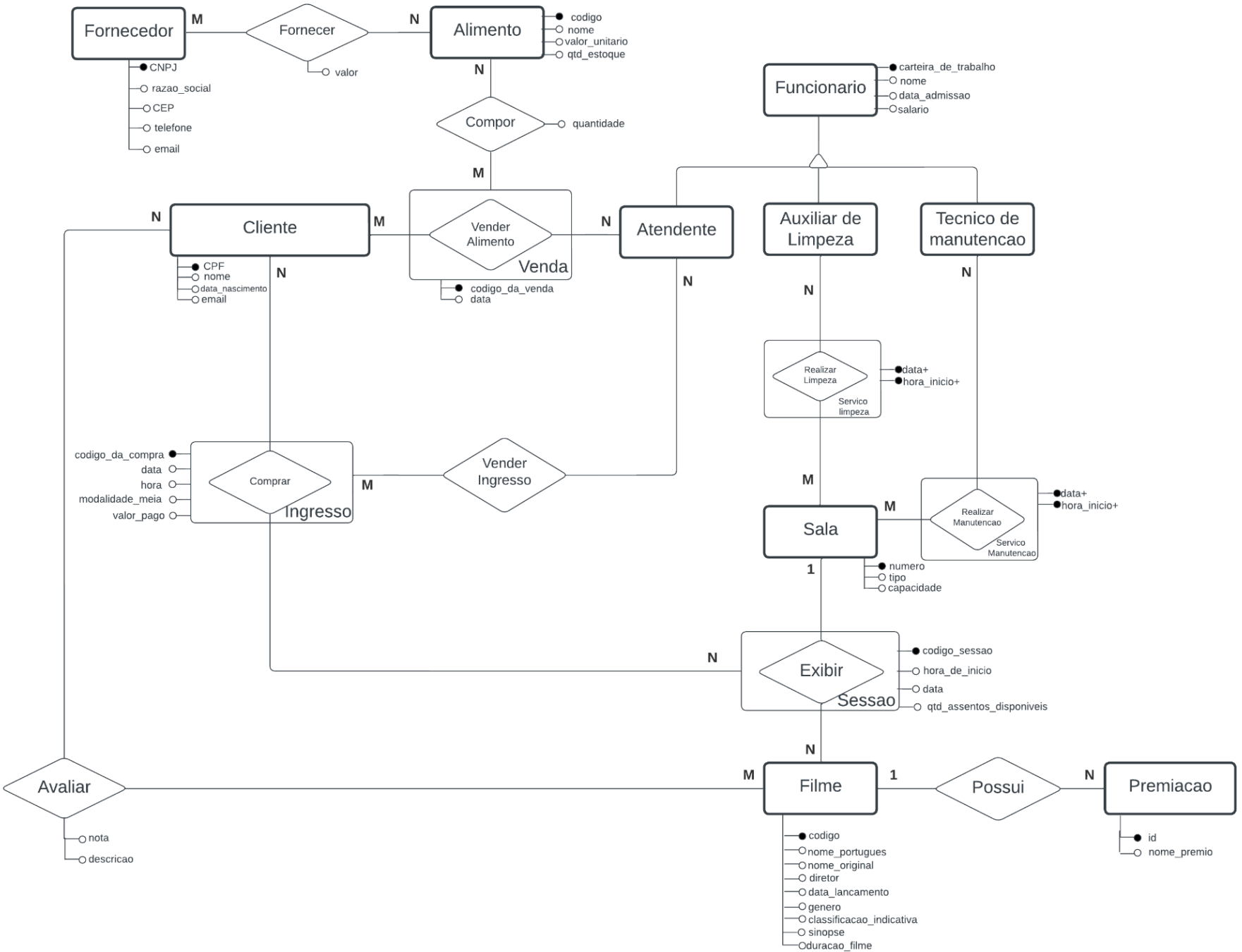
- Possui código, nome, valor unitário e quantidade em estoque.

1.2 Funcionalidades a serem atendidas

- Permitir a compra de ingressos para os filmes em cartaz no cinema, com informações sobre sala, lugar e sessão;
- Possibilitar a seleção de assentos nas salas de cinema;
- Permitir a compra de alimentos e bebidas, somente presencialmente e intermediado por um atendente;
- Exibir informações sobre os filmes, como sinopse, diretor, classificação indicativa e sessões disponíveis;
- Possibilitar opções de pagamento seguras para a compra de ingressos e alimentos no cinema;
- Disponibilizar informações sobre preços de produtos e ingressos no local;
- Possibilitar que usuários cadastrados façam avaliações sobre os filmes de 0 a 5 estrelas;
- Permitir o cadastro de novos usuários;
- Permitir o cadastro de novos fornecedores;
- Permitir o cadastro de novos funcionários.
- Permitir que os administradores do cinema programem sessões de filmes, definindo horários de início, datas e salas.
- Gerar relatórios sobre a bilheteria obtida da estreia de um determinado filme;
- Gerar relatórios quanto a disponibilidade dos alimentos na Bombonière;
- Gerar relatórios sobre os dias mais movimentados no cinema, a fim de auxiliar na decisão de promoções;
- Gerar relatórios sobre as preferências de filmes dos clientes, o que pode auxiliar na recomendação de outros semelhantes ou personalizar a seleção de filmes.

2. Projeto Conceitual do Banco de Dados

[Diagrama Entidade Relacionamento]



3. Projeto Lógico do Banco de Dados

Fornecedor = {CNPJ, razão_social, CEP, telefone, email}
Alimento = {codigo, nome, valor_unitario, qtd_estoque}
Fornece = {CNPJ, codigoAlimento, valor}
CNPJ - chave estrangeira que referencia Fornecedor
codigoAlimento - chave estrangeira que referencia Alimento
Funcionario = {carteira_de_trabalho, nome, data_admissao, salario, tipo}
onde tipo 1 - Atendente, 2 - Auxiliar de Limpeza, 3 - Tecnico de Manutencao.
Atendente = {carteira_de_trabalho}
carteira_de_trabalho - chave estrangeira que referencia Funcionario
Auxiliar de Limpeza = {carteira_de_trabalho}
carteira_de_trabalho - chave estrangeira que referencia Funcionario
Tecnico de Manutencao = {carteira_de_trabalho}
carteira_de_trabalho - chave estrangeira que referencia Funcionario
Sala = {numero, tipo, capacidade}
ServicoLimpeza = {carteira_de_trabalho, numeroSala, data, hora_inicio}
carteira_de_trabalho - chave estrangeira que referencia Auxiliar de Limpeza
numeroSala - chave estrangeira que referencia Sala
ServicoManutencao={carteira_de_trabalho, numeroSala, data, hora_inicio}
carteira_de_trabalho - chave estrangeira que referencia Tecnico de Manutencao
numeroSala - chave estrangeira que referencia Sala
Filme = {codigo, nome_portugues, nome_original, diretor, data_lancamento, genero, classificacao_indicativa, sinopse, premiacao, duracao_filme}
Cliente = {CPF, nome, data_nascimento, email}
Avaliação = {CPF_cliente, codigoFilme, nota, descricao}
CPF_cliente - chave estrangeira que referencia Cliente
codigoFilme - chave estrangeira que referencia Filme
Venda = {codigo_da_venda, carteira_de_trabalho, CPF_cliente, data}
carteira_de_trabalho - chave estrangeira que referencia Atendente
CPF_cliente - chave estrangeira que referencia Cliente
Compoe = {codigoAlimento, codigoVenda, quantidade}
codigoAlimento - chave estrangeira que referencia Alimento
codigoVenda - chave estrangeira que referencia Venda
Sessao = {codigo_sessao, numeroSala, codigoFilme, hora_de_inicio, data, qtd_assentos_disponiveis}
numeroSala - chave estrangeira que referencia Sala
codigoFilme - chave estrangeira que referencia Filme
numeroSala, hora_de_inicio, data - chave secundaria
Ingresso = {codigo_da_compra, CPF_cliente, codigoSessao, data, hora, modalidade_meia, valor_pago}
CPF_cliente - chave estrangeira que referencia Cliente
codigoSessao - chave estrangeira que referencia Sessao

Venda Ingresso = {carteira_de_trabalho, codigoCompra}
carteira_de_trabalho - chave estrangeira que referencia Atendente
codigoCompra - chave estrangeira que referencia Ingresso
Premiacao = {id, nome_premio, codigoFilme}
codigoFilme - chave estrangeira que referencia Filme

4. Projeto Físico do Banco de Dados

```
use ProjetoCinema
CREATE TABLE Fornecedor (
  CNPJ CHAR(14) PRIMARY KEY NOT NULL,
  razao_social CHAR(40) NOT NULL,
  CEP CHAR(8) NOT NULL,
  telefone NUMERIC(14, 0) NOT NULL,
  email VARCHAR(80) NOT NULL
);

CREATE TABLE Alimento (
  codigo NUMERIC(6,0) PRIMARY KEY NOT NULL,
  nome CHAR(40) NOT NULL,
  valor_unitario NUMERIC(7,2) NOT NULL,
  qtd_estoque INT NOT NULL
);

CREATE TABLE Fornece (
  CNPJ CHAR(14) NOT NULL,
  codigoAlimento NUMERIC(6,0) NOT NULL,
  valor NUMERIC(10, 2) NOT NULL,

  PRIMARY KEY (CNPJ, codigoAlimento),
  FOREIGN KEY (CNPJ) REFERENCES Fornecedor (CNPJ),
  FOREIGN KEY (codigoAlimento) REFERENCES Alimento (codigo)
);

CREATE TABLE Funcionario (
  carteira_de_trabalho NUMERIC(11,0) PRIMARY KEY NOT NULL,
  nome CHAR(50) NOT NULL,
  data_admissao DATE NOT NULL,
  salario NUMERIC(10, 2) NOT NULL,
  tipo INT NOT NULL
);
```

```sql
CREATE TABLE Atendente (
  carteira_de_trabalho NUMERIC(11,0) PRIMARY KEY NOT NULL,
  FOREIGN KEY (carteira_de_trabalho) REFERENCES Funcionario
(carteira_de_trabalho)
);

CREATE TABLE Auxiliar_de_Limpeza (
  carteira_de_trabalho NUMERIC(11,0) PRIMARY KEY NOT NULL,
  FOREIGN KEY (carteira_de_trabalho) REFERENCES Funcionario
(carteira_de_trabalho)
);

CREATE TABLE Tecnico_de_Manutencao (
  carteira_de_trabalho NUMERIC(11,0) PRIMARY KEY NOT NULL,
  FOREIGN KEY (carteira_de_trabalho) REFERENCES Funcionario
(carteira_de_trabalho)
);

CREATE TABLE Sala (
  numero INT PRIMARY KEY NOT NULL,
  tipo CHAR(20) NOT NULL,
  capacidade INT NOT NULL,
  -- removido assento_disponivel INT NOT NULL e colocado em Sessao
);

CREATE TABLE ServicoLimpeza (
  carteira_de_trabalho NUMERIC(11,0) NOT NULL,
  numeroSala INT NOT NULL,
  data DATE NOT NULL,
  hora_inicio TIME NOT NULL,

  PRIMARY KEY (carteira_de_trabalho, numeroSala, data, hora_inicio),
  FOREIGN KEY (carteira_de_trabalho) REFERENCES Auxiliar_de_Limpeza
(carteira_de_trabalho),
  FOREIGN KEY (numeroSala) REFERENCES Sala (numero)
);

CREATE TABLE Servico_Manutencao (
  carteira_de_trabalho NUMERIC(11,0) NOT NULL,
  numeroSala INT NOT NULL,
  data DATE NOT NULL,
  hora_inicio TIME NOT NULL,
```

```
  PRIMARY KEY (carteira_de_trabalho, numeroSala, data, hora_inicio),
  FOREIGN KEY (carteira_de_trabalho) REFERENCES Tecnico_de_Manutencao
(carteira_de_trabalho),
  FOREIGN KEY (numeroSala) REFERENCES Sala (numero)
);

CREATE TABLE Filme (
  codigo NUMERIC(6,0) PRIMARY KEY NOT NULL,
  nome_portugues VARCHAR(80) NOT NULL,
  nome_original VARCHAR(80) NOT NULL,
  diretor VARCHAR(80) NOT NULL,
  data_lancamento DATE NOT NULL,
  genero VARCHAR(30) NOT NULL,
  classificacao_indicativa VARCHAR(10) NOT NULL,
  sinopse VARCHAR(200) NOT NULL,
  duracao_filme INT NOT NULL
);

CREATE TABLE Cliente (
  CPF CHAR(11) PRIMARY KEY NOT NULL,
  nome VARCHAR(100) NOT NULL,
  data_nascimento DATE NOT NULL,
  email VARCHAR(100) NOT NULL
);

CREATE TABLE Avaliacao (
  CPF_cliente CHAR(11) NOT NULL,
  codigoFilme NUMERIC(6,0) NOT NULL,
  nota NUMERIC(2, 1) NOT NULL,
  descricao VARCHAR(400),

  PRIMARY KEY (CPF_cliente, codigoFilme),
  FOREIGN KEY (CPF_cliente) REFERENCES Cliente (CPF),
  FOREIGN KEY (codigoFilme) REFERENCES Filme (codigo)
);

CREATE TABLE VendaAlimento (
  codigo_da_venda NUMERIC(6,0) PRIMARY KEY NOT NULL,
  carteira_de_trabalho NUMERIC(11,0) NOT NULL,
  CPF_cliente CHAR(11) NOT NULL,
  data DATETIME NOT NULL,
  FOREIGN KEY (carteira_de_trabalho) REFERENCES Atendente
(carteira_de_trabalho),
```

```
  FOREIGN KEY (CPF_cliente) REFERENCES Cliente (CPF)
);


CREATE TABLE Compoe (
  codigoAlimento NUMERIC(6,0) NOT NULL,
  codigoVenda NUMERIC(6,0) NOT NULL,
  quantidade NUMERIC(3, 0) NOT NULL,

  PRIMARY KEY(codigoAlimento, codigoVenda),
  FOREIGN KEY (codigoAlimento) REFERENCES Alimento (codigo),
  FOREIGN KEY (codigoVenda) REFERENCES VendaAlimento (codigo_da_venda)
);

CREATE TABLE Sessao (
  codigo_sessao NUMERIC(6,0) NOT NULL,
  numeroSala INT NOT NULL,
  codigoFilme NUMERIC(6,0) NOT NULL,
  hora_de_inicio TIME NOT NULL CHECK (hora_de_inicio IN ('14:00',
'14:30', '16:30', '18:00', '19:30', '20:00', '22:00', '00:00')),
  data DATE NOT NULL,
  qtd_assentos_disponiveis INT NOT NULL,


  PRIMARY KEY(codigo_sessao),
  UNIQUE(numeroSala, hora_de_inicio, data),
  FOREIGN KEY (numeroSala) REFERENCES Sala (numero),
  FOREIGN KEY (codigoFilme) REFERENCES Filme (codigo)
);

CREATE TABLE Ingresso (
  codigo_da_compra NUMERIC(6,0) PRIMARY KEY NOT NULL,
  CPF_cliente CHAR(11) NOT NULL,
  codigoSessao NUMERIC(6,0) NOT NULL,
  data DATE NOT NULL,
  hora TIME NOT NULL,
  modalidade_meia BIT NOT NULL,
  valor_pago DECIMAL(10, 2) NOT NULL,
  FOREIGN KEY (CPF_cliente) REFERENCES Cliente (CPF),
  FOREIGN KEY (codigoSessao) REFERENCES Sessao (codigo_sessao)
);

CREATE TABLE VendaIngresso (
```

```
  carteira_de_trabalho NUMERIC(11,0) NOT NULL,
  codigoCompra NUMERIC(6,0) NOT NULL,

  PRIMARY KEY(carteira_de_trabalho, codigoCompra),
  FOREIGN KEY (carteira_de_trabalho) REFERENCES Atendente
(carteira_de_trabalho),
  FOREIGN KEY (codigoCompra) REFERENCES Ingresso (codigo_da_compra)
);

CREATE TABLE Premiacao (
id int IDENTITY(1,1) NOT NULL PRIMARY KEY,
nome_premio varchar(80) NOT NULL,
codigoFilme numeric(6,0) NOT NULL,

FOREIGN KEY (codigoFilme) REFERENCES Filme
)


/*Criando indices somente para chaves estrangeiras*/
CREATE INDEX idx_fk_carteira_trabalho_venda_alimento
ON VendaAlimento (carteira_de_trabalho);

CREATE INDEX idx_fk_cpfcli_venda_alimento
ON VendaAlimento (CPF_cliente);

CREATE INDEX idx_fk_numero_sala_sessao
ON Sessao (numeroSala);

CREATE INDEX idx_fk_codigo_filme_sessao
ON Sessao (codigoFilme);

CREATE INDEX idx_fk_codigoFilme_premiacao
ON Premiacao (codigoFilme);

CREATE INDEX idx_fk_cpfcli_ingresso
ON Ingresso (CPF_Cliente);

CREATE INDEX idx_fk_codigosessao_ingresso
ON Ingresso (codigoSessao);
```

5. Manipulação do Banco de Dados

- **Visões**

```
/*Obtem informações sobre as sessoes do cinema*/
CREATE VIEW vw_SessoesFilmes AS
SELECT S.codigo_sessao, S.numeroSala, S.codigoFilme, S.hora_de_inicio,
S.data,
       SA.tipo AS tipo_sala, F.nome_portugues AS nome_filme
FROM Sessao S
JOIN Sala SA ON S.numeroSala = SA.numero
JOIN Filme F ON S.codigoFilme = F.codigo

go
/*Obtem alimentos na bomboniere e informa o Fornecedor que realizou
esse abastecimento*/
CREATE VIEW vw_AlimentosEstoque AS
SELECT A.codigo, A.nome, A.valor_unitario, A.qtd_estoque,
forn.razao_social AS 'Fornecido por'
FROM Alimento A
LEFT JOIN Fornece F ON A.codigo = F.codigoAlimento
LEFT JOIN Fornecedor forn ON F.CNPJ = forn.CNPJ

go
/*Obtem filmes assistidos pelos clientes*/
CREATE VIEW vw_ClientesCompras AS
SELECT MIN(I.codigo_da_compra) codigo_da_compra, I.CPF_cliente, C.nome
AS nome_cliente, I.codigoSessao,
       F.nome_portugues AS filme_assistido
FROM Ingresso I
JOIN Cliente C ON I.CPF_cliente = C.CPF
JOIN Sessao S  ON S.codigo_sessao = I.codigoSessao
JOIN Filme F   ON F.codigo = S.codigoFilme
GROUP BY I.CPF_cliente, C.nome, I.codigoSessao, F.nome_portugues


go
/*Apresenta lucro obtido dos filmes*/
CREATE VIEW visao_filme_e_lucroObtido
AS
SELECT Sessao.codigoFilme, SUM (Ingresso.valor_pago) 'Lucro obtido'
FROM Sessao INNER JOIN Ingresso
```

```
    ON Ingresso.codigoSessao = Sessao.codigo_sessao
GROUP BY Sessao.codigoFilme

go
/*Obtem todos filmes com suas medias de avaliacao*/
CREATE VIEW visao_nota_media_filme
AS
SELECT Avaliacao.codigoFilme, Filme.nome_portugues, AVG(Avaliacao.nota)
'Nota média'
FROM Avaliacao INNER JOIN Filme
    on Avaliacao.codigoFilme = Filme.codigo
GROUP BY Avaliacao.codigoFilme, Filme.nome_portugues


go
/*Obtem funcionarios do Cinema, incluindo sua respectiva funcao.*/
CREATE VIEW vw_InformacoesFuncionario AS
SELECT F.carteira_de_trabalho, F.nome, F.data_admissao, F.salario,
F.tipo,
       A.carteira_de_trabalho AS Atendente,
       AL.carteira_de_trabalho AS 'Auxiliar de Limpeza',
       TM.carteira_de_trabalho AS 'Técnico de Manutenção'
FROM Funcionario F
LEFT JOIN Atendente A ON F.carteira_de_trabalho =
A.carteira_de_trabalho
LEFT JOIN Auxiliar_de_Limpeza AL ON F.carteira_de_trabalho =
AL.carteira_de_trabalho
LEFT JOIN Tecnico_de_Manutencao TM ON F.carteira_de_trabalho =
TM.carteira_de_trabalho


go
/*Obtem serviços de manutençao e carteira do técnico responsável*/
CREATE VIEW vw_ServicosManutencaoSala AS
SELECT SM.carteira_de_trabalho, SM.numeroSala, SM.data, SM.hora_inicio,
       S.tipo AS tipo_sala, S.capacidade
FROM Servico_Manutencao SM
JOIN Sala S ON SM.numeroSala = S.numero


go
```

```sql
/*Obtem serviços de limpeza e a carteira do funcionário que a
realizou*/
CREATE VIEW vw_ServicosLimpezaSala AS
SELECT SL.carteira_de_trabalho, SL.numeroSala, SL.data, SL.hora_inicio,
       S.tipo AS tipo_sala, S.capacidade
FROM ServicoLimpeza SL
JOIN Sala S ON SL.numeroSala = S.numero


go
/*Obtém informações dos fornecedores que já realizaram entregas para o
cinema*/
CREATE VIEW vw_InformacoesFornecedor AS
SELECT F.CNPJ, F.razao_social, F.CEP, F.telefone, F.email,
       FF.codigoAlimento, FF.valor
FROM Fornecedor F
JOIN Fornece FF ON F.CNPJ = FF.CNPJ
```

- **Procedimentos Armazenados**

```
/*Cadastro de clientes*/
CREATE PROCEDURE cadastrar_cliente
@cpf char(11),
@nome_cliente varchar(100),
@data_nascimento date,
@email varchar(100)
AS
INSERT INTO Cliente
VALUES (@cpf, @nome_cliente, @data_nascimento, @email)

go
/*Cadastro de Funcionarios*/
CREATE PROCEDURE cadastrar_funcionario
@carteira_trabalho numeric(11,0),
@nome_funcionario char(50),
@data_admissao date,
@salario numeric(10,2),
@tipo int -- 1 para Atendente, 2 para Auxiliar de Limpeza, 3 para
Técnico de Manutenção
AS
BEGIN TRAN
INSERT INTO Funcionario
VALUES (@carteira_trabalho, @nome_funcionario, @data_admissao,
@salario, @tipo)
IF @@ROWCOUNT > 0
BEGIN
    IF @tipo = 1
    BEGIN
        INSERT INTO Atendente
        VALUES (@carteira_trabalho)
        COMMIT
        RETURN 0
    END
    ELSE IF @tipo = 2
    BEGIN
        INSERT INTO Auxiliar_de_Limpeza
        VALUES (@carteira_trabalho)
        COMMIT
        RETURN 0
    END
    ELSE IF @tipo = 3
```

```
    BEGIN
        INSERT INTO Tecnico_de_Manutencao
        VALUES (@carteira_trabalho)
        COMMIT
        RETURN 0
    END

    ELSE
    BEGIN
        PRINT 'Tipo de funcionario invalido';
        ROLLBACK
        RETURN 1
    END
END
ELSE
BEGIN
    ROLLBACK
    RETURN 1
END

go
/*Cadastro de salas do cinema*/
CREATE PROCEDURE cadastrar_sala
@numero_sala int,
@tipo char(20),
@capacidade int
AS
INSERT INTO Sala
VALUES (@numero_sala, @tipo, @capacidade)

go
/*Registro de Servico_Limpeza.PRÉ-REQUISITO: Auxiliar_de_Limpeza e Sala
já inseridos no banco*/
CREATE PROCEDURE registrar_limpeza_realizada
@carteira_trabalho_limpeza numeric(11,0),
@numeroSala int,
@data_limpeza date,
@hora_limpeza time
AS

INSERT INTO ServicoLimpeza
VALUES (@carteira_trabalho_limpeza, @numeroSala, @data_limpeza,
@hora_limpeza)
```

```
go
/*Busca salas que não receberam limpezas em uma determinada data*/
CREATE PROCEDURE salas_nao_limpas_na_data
@data_desejada date = NULL
AS
IF @data_desejada = NULL
BEGIN
    SET @data_desejada = GETDATE()
END
SELECT Sala.numero, ServicoLimpeza.numeroSala
FROM Sala LEFT JOIN ServicoLimpeza
    ON Sala.numero = ServicoLimpeza.numeroSala AND ServicoLimpeza.data
= @data_desejada
WHERE ServicoLimpeza.numeroSala IS NULL

go
/*Registro de Servico_Manutencao.PRÉ-REQUISITO: Tecnico_de_Manutencao e
Sala já inseridos no banco*/
CREATE PROCEDURE registrar_manutencao_realizada
@carteira_trabalho_tecnico numeric(11,0),
@numeroSala int,
@data_manutencao date,
@hora_manutencao time
AS
INSERT INTO Servico_Manutencao
VALUES (@carteira_trabalho_tecnico, @numeroSala, @data_manutencao,
@hora_manutencao)

go
/*Cadastro de fornecedores de alimentos*/
CREATE PROCEDURE cadastro_fornecedor
@cnpj char(14),
@razao_social char(40),
@cep char(8),
@telefone numeric(14,0),
@email varchar(80)
AS
INSERT INTO Fornecedor
VALUES (@cnpj, @razao_social, @cep, @telefone, @email)
```

```
go
/*Inserção de alimentos no estoque do cinema*/
CREATE PROCEDURE cadastrar_alimento
@codigo_alimento numeric(6,0),
@nome_alimento char(40),
@valor_unitario numeric(7,2),
@qtd_estoque int
AS
INSERT INTO Alimento
VALUES (@codigo_alimento, @nome_alimento, @valor_unitario,
@qtd_estoque)

go
/*Fornecimento de alimentos e atualização do estoque. PRÉ-REQUISITO:
Fornecedor e Alimento já inseridos no banco*/
CREATE PROCEDURE fornece_alimento
@cnpj_fornecedor char(14),
@codigoAlimento numeric(6,0),
@valor_fornecimento numeric(10,2),
@qtd_fornecida int
AS
BEGIN TRAN
IF @qtd_fornecida <= 0
BEGIN
    PRINT 'Quantidade fornecida inválida.'
    ROLLBACK
    RETURN 1
END
ELSE
BEGIN
    INSERT INTO Fornece
    VALUES (@cnpj_fornecedor, @codigoAlimento, @valor_fornecimento)
    IF @@ROWCOUNT > 0
    BEGIN
        UPDATE Alimento
        SET qtd_estoque = qtd_estoque + @qtd_fornecida
        WHERE codigo = @codigoAlimento
        IF @@ROWCOUNT > 0
        BEGIN
            COMMIT
            RETURN 0
        END
        ELSE
```

```
        BEGIN
            ROLLBACK
            RETURN 1
        END
    END
    ELSE
    BEGIN
        ROLLBACK
        RETURN 1
    END
END

go
/*Cadastrar venda de alimento. PRÉ-REQUISITO: Cliente e Atendente já
inseridos no banco.*/
CREATE PROCEDURE cadastrar_vendaAlimento
@codigo_da_venda numeric(6,0),
@carteira_trabalho_atendente numeric(11,0),
@cpf_cliente char(11),
@data_venda datetime
AS
INSERT INTO VendaAlimento
VALUES (@codigo_da_venda, @carteira_trabalho_atendente, @cpf_cliente,
@data_venda)

go
/*Verificar a disponibilidade de determinado alimento no estoque.
PRÉ-REQUISITO: Alimento já inserido no banco*/
CREATE PROCEDURE verifica_estoque_disponivel
@codigoAlimento numeric(6,0),
@quantidade_solicitada numeric(3,0)
AS
DECLARE @quantidade_disponivel numeric(3,0)
DECLARE @nome_alimento char(40)

SELECT @quantidade_disponivel = qtd_estoque, @nome_alimento = nome
FROM Alimento
WHERE codigo = @codigoAlimento
IF @quantidade_solicitada > @quantidade_disponivel
BEGIN
    DECLARE @alerta_estoque varchar(200)
    SET @alerta_estoque = 'Estoque insuficiente de: ' + @nome_alimento
    RAISERROR(@alerta_estoque, 16, 1)
```

```
    RETURN 1
END
ELSE
BEGIN
    IF @quantidade_disponivel < 10
    BEGIN
        PRINT 'Estoque proximo do fim: ' + @nome_alimento
    END

    RETURN 0
END

go
/*Cadastrar itens de uma determinada venda de alimentos e atualizar o
estoque. PRÉ-REQUISITO: Venda e Alimento já inseridos no banco;
PROCEDURE verifica_estoque_disponivel já criado.*/
CREATE PROCEDURE cadastrar_compoeVenda
@codigoAlimento numeric(6,0),
@codigoVenda numeric(6,0),
@quantidade_vendida numeric(3,0)
AS
BEGIN TRAN
DECLARE @retorno int -- 0 => disponivel, 1 => estoque em falta
EXEC @retorno = verifica_estoque_disponivel @codigoAlimento,
@quantidade_vendida
IF @retorno = 0
BEGIN
    INSERT INTO Compoe
    VALUES (@codigoAlimento, @codigoVenda, @quantidade_vendida)
    IF @@ROWCOUNT > 0
    BEGIN
        UPDATE Alimento
        SET qtd_estoque = qtd_estoque - @quantidade_vendida
        WHERE codigo = @codigoAlimento
        IF @@ROWCOUNT > 0
        BEGIN
            COMMIT
            RETURN 0
        END
        ELSE
        BEGIN
            ROLLBACK
            RETURN 1
```

```
        END
    END
    ELSE
    BEGIN
        ROLLBACK
        RETURN 1
    END
END
ELSE
BEGIN
    ROLLBACK
    RETURN 1
END

go
/*Inserção de filmes no cartaz*/
CREATE PROCEDURE cadastrar_filme
@codigo_filme numeric(6,0),
@nome_portugues varchar(80),
@nome_original varchar(80),
@diretor varchar(80),
@data_lancamento date,
@genero varchar(30),
@classificacao_indicativa varchar(10),
@sinopse varchar(200),

@duracao_minutos_filme int
AS
INSERT INTO Filme
VALUES (@codigo_filme, @nome_portugues, @nome_original, @diretor,
@data_lancamento, @genero, @classificacao_indicativa, @sinopse,
@premiacao, @duracao_minutos_filme)

go
/*Cadastro de Sessões no cinema. PRÉ-REQUISITO: ter Filme e Sala já
inseridos no banco*/
CREATE PROCEDURE cadastrar_sessao
@codigo_sessao numeric(6,0),
@numeroSala int,
@codigoFilme numeric(6,0),
@hora_de_inicio time,
@data date,
@qtd_assentos_disponiveis int
```

```
AS
BEGIN TRAN
DECLARE @capacidade_sala int
SELECT @capacidade_sala = capacidade
FROM Sala
WHERE numero = @numeroSala

IF @qtd_assentos_disponiveis > @capacidade_sala
BEGIN
    RAISERROR('Assentos disponiveis para a Sessao excedeu a capacidade
maxima da Sala escolhida.', 16, 1)
    ROLLBACK
    RETURN 1
END
ELSE
BEGIN
    INSERT INTO Sessao
    VALUES (@codigo_sessao, @numeroSala, @codigoFilme, @hora_de_inicio,
@data, @qtd_assentos_disponiveis)
    IF @@ROWCOUNT > 0
    BEGIN
        COMMIT
        RETURN 0
    END
    ELSE
    BEGIN
        ROLLBACK
        RETURN 1
    END
END


go
/*verificando assentos ainda disponiveis em uma determinada sessao.
PRÉ-REQUISITO: Sessao já inserida no banco*/
CREATE PROCEDURE verifica_assentos_sessao
@codigoSessao numeric (6,0)
AS
DECLARE @qtd_assentos_livres int
SELECT @qtd_assentos_livres = qtd_assentos_disponiveis
FROM Sessao
WHERE codigo_sessao = @codigoSessao
```

```
RETURN @qtd_assentos_livres

go
/*Compra de ingresso feita pelo Cliente (não intermediada por
atendente), verificando disponibilidade da sessão e atualizando os
lugares disponíveis nessa determinada sessão.
PRÉ-REQUISITO: Cliente e Sessão já inseridos no banco*/
CREATE PROCEDURE compra_ingresso_cliente
@codigo_da_compra numeric(6,0),
@CPF_cliente char(11),
@codigoSessao numeric(6,0),
@data date,
@hora time,
@modalidade_meia bit,
@valor_pago decimal(10,2)
AS
BEGIN TRAN
DECLARE @qtd_assentos_disponiveis int
exec @qtd_assentos_disponiveis = verifica_assentos_sessao @codigoSessao

IF @qtd_assentos_disponiveis > 0
BEGIN
    IF @modalidade_meia = 1
    BEGIN
        SET @valor_pago = @valor_pago / 2
    END

    INSERT INTO Ingresso
    VALUES (@codigo_da_compra, @CPF_cliente, @codigoSessao, @data,
@hora, @modalidade_meia, @valor_pago)

    IF @@ROWCOUNT > 0
    BEGIN --atualizar assentos disponiveis na sessao
        UPDATE Sessao
        SET qtd_assentos_disponiveis = qtd_assentos_disponiveis - 1
        WHERE codigo_sessao = @codigoSessao
        IF @@ROWCOUNT > 0
        BEGIN
            COMMIT
            RETURN 0
        END
        ELSE
        BEGIN
```

```
            ROLLBACK
            RETURN 1
        END
    END
    ELSE
    BEGIN
        ROLLBACK
        RETURN 1
    END
END
ELSE
BEGIN
    RAISERROR('Sessao esgotada', 16, 1)
    ROLLBACK
    RETURN 1
END


go
/*Associar uma compra de ingresso a um atendente que realizou essa
venda presencialmente. PRÉ-REQUISITO: Atendente e Ingresso já inseridos
no banco*/
CREATE PROCEDURE registra_venda_ingresso
@carteira_de_trabalho_atendente numeric(11,0),
@codigoCompra numeric(6,0)
AS
INSERT INTO VendaIngresso
VALUES (@carteira_de_trabalho_atendente, @codigoCompra)

go
/*Registro de avaliação feita por um cliente de um determinado filme.
PRÉ-REQUISITO: Cliente e Filme já inseridos no banco*/
CREATE PROCEDURE cadastrar_avaliacao_filme
@cpf char(11),
@codigoFilme_avaliado numeric(6,0),
@nota numeric(2,1),
@descricao varchar(400)
AS
INSERT INTO Avaliacao
VALUES (@cpf, @codigoFilme_avaliado, @nota, @descricao)
```

```
go
/*Informar a media de um determinado filme no banco.
PRÉ-REQUISITO: visao_nota_media_filme já criada no banco; Avaliação já
inserida no banco.*/
CREATE PROCEDURE busca_media_filme
@nome_filme_desejado varchar(80)= '%'
AS
SET @nome_filme_desejado = '%' + @nome_filme_desejado + '%';
SELECT *
FROM visao_nota_media_filme
WHERE nome_portugues like @nome_filme_desejado


go
/*Informar dia de maior venda de ingressos no intervalo desejado*/
CREATE PROCEDURE dia_maior_venda_mes
@data_inicio DATE,
@data_fim DATE
AS
SELECT TOP 1 data, SUM(valor_pago) 'Lucro no dia'
FROM Ingresso
WHERE data BETWEEN @data_inicio and @data_fim
GROUP BY data
ORDER BY 'Lucro no dia' desc

go
/*Informar os filmes assistidos por um determinado cliente.
PRÉ-REQUISITO: vw_ClientesCompras já criada no banco.*/
CREATE PROCEDURE filme_assistido_cliente
@nome_desejado varchar(100) = '%'
AS
SELECT CPF_cliente, nome_cliente, filme_assistido
FROM vw_ClientesCompras
WHERE nome_cliente like '%' + @nome_desejado + '%'


go
/*Busca bilheteria do filme na estreia*/
CREATE PROCEDURE busca_estreia_filme
@codigo_filme numeric(6,0)
AS
SELECT Sessao.codigoFilme, Filme.nome_portugues, SUM
(Ingresso.valor_pago) 'Lucro obtido'
```

```
FROM Sessao INNER JOIN Ingresso
    ON Ingresso.codigoSessao = Sessao.codigo_sessao
    INNER JOIN Filme
    ON Filme.codigo = Sessao.codigoFilme
WHERE Filme.codigo = @codigo_filme and Ingresso.data = data_lancamento
GROUP BY Sessao.codigoFilme, Filme.nome_portugues

go
/*Busca bilheteria total do filme*. PRÉ-REQUISITO:
visao_filme_e_lucroObtido já criada no banco.*/
CREATE PROCEDURE busca_lucroTotal_filme
@codigoFilme numeric(6,0)
AS
SELECT codigoFilme, Filme.nome_portugues, [Lucro obtido]
FROM visao_filme_e_lucroObtido INNER JOIN Filme
    ON visao_filme_e_lucroObtido.codigoFilme = Filme.codigo
WHERE Filme.codigo = @codigoFilme


go
```

- **Gatilhos**

```
/*Para cada vez que for alterada a quantidade de um alimento que compoe
a venda, atualizar o estoque. PRÉ-REQUISITO: inserção do alimento já
feita, através do procedure cadastrar_compoeVenda*/
CREATE TRIGGER compoe_ao_atualizar
ON Compoe
FOR UPDATE
AS
IF UPDATE(quantidade)
BEGIN
    UPDATE Alimento
    SET qtd_estoque = qtd_estoque - (SELECT inserted.quantidade -
deleted.quantidade
                                     FROM inserted INNER JOIN deleted
                                         ON inserted.codigoAlimento =
deleted.codigoAlimento
                                         AND inserted.codigoVenda =
deleted.codigoVenda
                                     )
    WHERE Alimento.codigo = (SELECT codigoAlimento FROM inserted)
    IF @@ROWCOUNT = 0
        ROLLBACK
END

go
/*Cada vez que uma nova avaliação for inserida, verificar se a nota
media desse filme é maior que a dos demais. Se sim, adicionar premio ao
filme chamado "Melhor avaliacao local"*/
CREATE TRIGGER indica_melhor_avaliacao_local
ON Avaliacao
AFTER INSERT
AS
BEGIN
    DECLARE @filme_recem_avaliado numeric(6,0) = (SELECT codigoFilme
FROM inserted)
    --condicoes para saber se o recem inserido tem melhor avaliacao
    IF (SELECT TOP 1 [Nota Média] FROM visao_nota_media_filme WHERE
codigoFilme = @filme_recem_avaliado ORDER BY [Nota Média] DESC)
    <= (SELECT [Nota Média] FROM visao_nota_media_filme WHERE
codigoFilme = @filme_recem_avaliado)
    AND NOT EXISTS (SELECT 1 FROM Premiacao WHERE codigoFilme =
@filme_recem_avaliado AND nome_premio = 'Melhor Avaliação Local')
```

```
    BEGIN
        --remover premio anterior, caso outro o possui, mas agora,
         --perdeu sua posicao
        DELETE FROM Premiacao WHERE nome_premio = 'Melhor Avaliação
Local'

        INSERT INTO Premiacao (nome_premio, codigoFilme)
        VALUES ('Melhor Avaliação Local', @filme_recem_avaliado)

        IF @@ROWCOUNT = 0
            ROLLBACK
    END

    --verificando a avaliacao recente foi sobre o filme que era o
--melhor, mas acabou abaixando sua media e o fez deixar de ser o melhor
    IF (SELECT TOP 1 [Nota Média] FROM visao_nota_media_filme ORDER BY
[Nota Média] DESC)
    > (SELECT [Nota Média] FROM visao_nota_media_filme WHERE
codigoFilme = @filme_recem_avaliado)
    AND EXISTS (SELECT 1 FROM Premiacao WHERE codigoFilme =
@filme_recem_avaliado AND nome_premio = 'Melhor Avaliação Local')
    BEGIN

        DELETE FROM Premiacao WHERE codigoFilme = @filme_recem_avaliado
AND nome_premio = 'Melhor Avaliação Local'

        --procurando o novo filme a receber o premio
        DECLARE @novo_melhor_filme numeric (6,0) = (SELECT TOP 1
codigoFilme
                                                    FROM
visao_nota_media_filme
                                                    ORDER BY [Nota
Média] DESC)


        --@novo_melhor_filme recebendo o premio
        INSERT INTO Premiacao (nome_premio, codigoFilme)
        VALUES ('Melhor Avaliação Local', @novo_melhor_filme)

        IF @@ROWCOUNT = 0
            ROLLBACK
    END
END
```