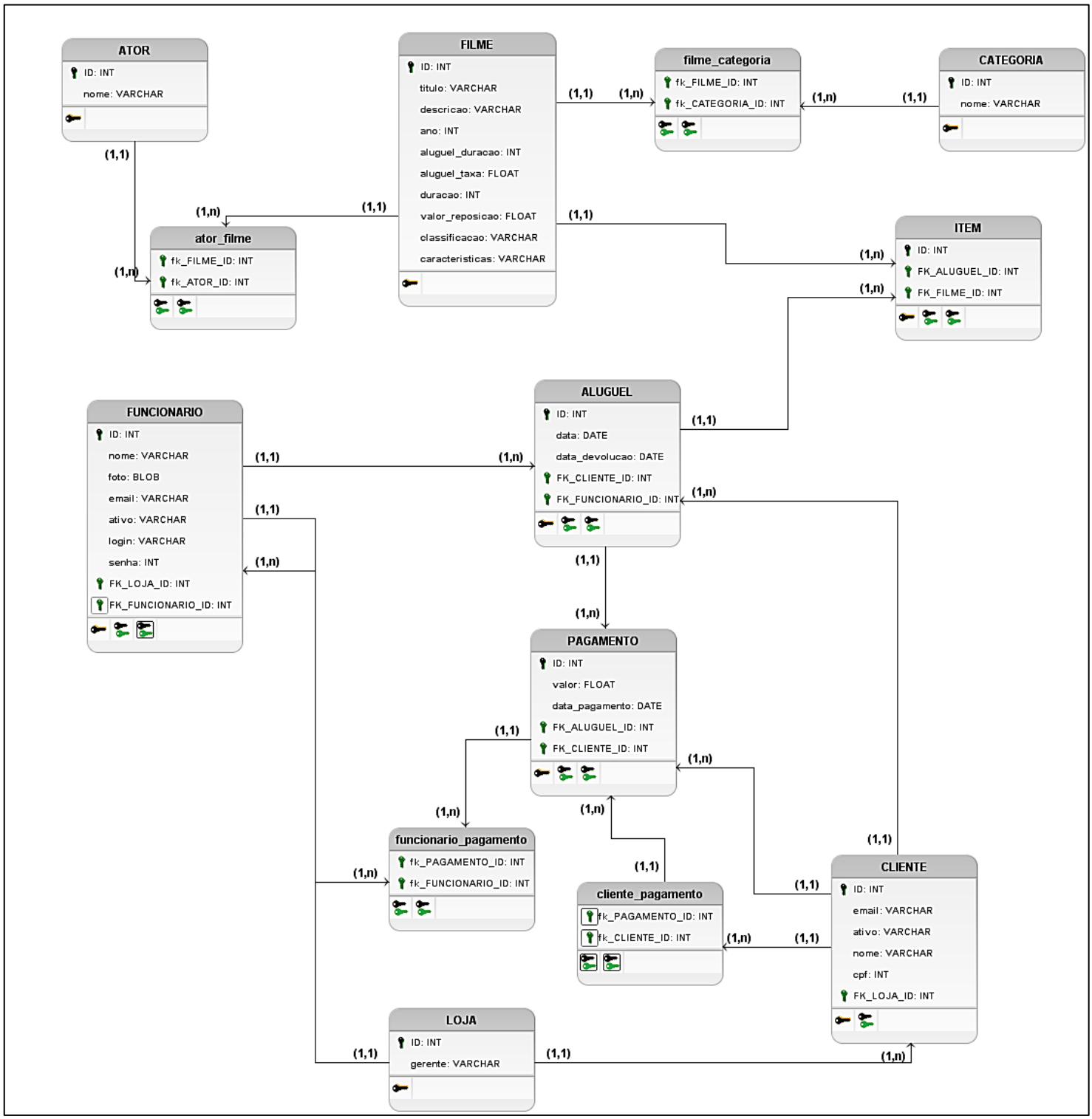
MAIKON FABRÍCIO GINO **A: 2901392313007**

01)
- Modelo lógico:



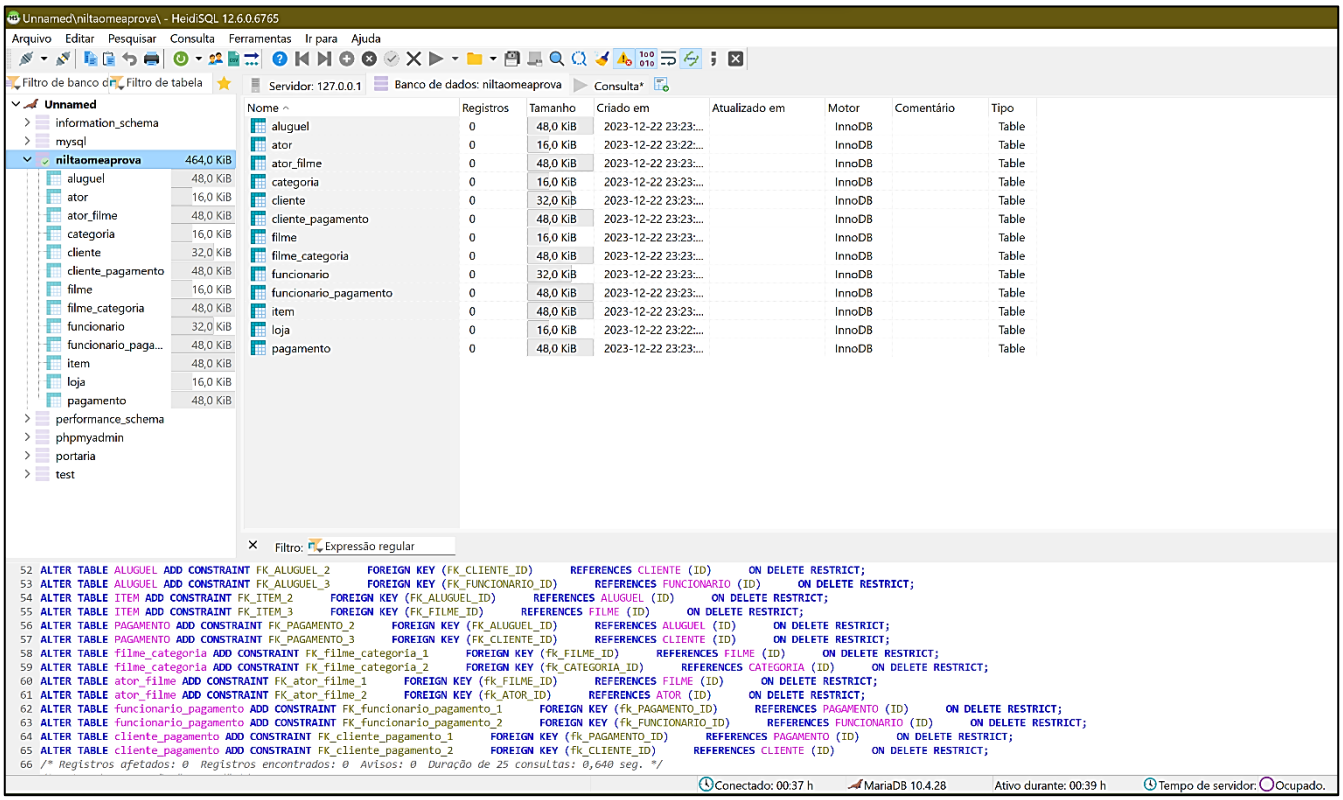
- Modelo Físico / Scripts:

```
CREATE DATABASE niltaomeaprova;

CREATE TABLE LOJA (
    ID INT PRIMARY KEY,
    gerente VARCHAR (50)
);

CREATE TABLE CLIENTE (
    ID INT PRIMARY KEY,
    email VARCHAR (50),
    ativo VARCHAR (50),
    nome VARCHAR (50),
    cpf INT,
    FK_LOJA_ID INT
);

CREATE TABLE FUNCIONARIO (
    ID INT PRIMARY KEY,
    nome VARCHAR (50),
    foto BLOB,
    email VARCHAR (50),
    ativo VARCHAR (50),
    login VARCHAR (50),    ID INT PRIMARY KEY,
    email VARCHAR (50),
    ativo VARCHAR (50),
    nome VARCHAR (50),
    cpf INT,
    FK_LOJA_ID INT

    senha INT,
    FK_LOJA_ID INT,
    FK_FUNCIONARIO_ID INT
);

CREATE TABLE ATOR (
    ID INT PRIMARY KEY,
    nome VARCHAR (50)
);

CREATE TABLE CATEGORIA (
    ID INT PRIMARY KEY,
    nome VARCHAR (50)
);

CREATE TABLE FILME (
```

```
    ID INT PRIMARY KEY,
    titulo VARCHAR (50),
    descricao VARCHAR (500),
    ano INT,
    aluguel_duracao INT,
    aluguel_taxa FLOAT,
    duracao INT,
    valor_reposicao FLOAT,
    classificacao INT ,
    caracteristicas VARCHAR (150)
);

CREATE TABLE ALUGUEL (
    ID INT PRIMARY KEY,
    data DATE,
    data_devolucao DATE,
    FK_CLIENTE_ID INT,
    FK_FUNCIONARIO_ID INT
);

CREATE TABLE ITEM (
    ID INT PRIMARY KEY,
    FK_ALUGUEL_ID INT,
    FK_FILME_ID INT
);

CREATE TABLE PAGAMENTO (
    ID INT PRIMARY KEY,
    valor FLOAT,
    data_pagamento DATE,
    FK_ALUGUEL_ID INT,
    FK_CLIENTE_ID INT
);

CREATE TABLE filme_categoria (
    fk_FILME_ID INT,
    fk_CATEGORIA_ID INT
);

CREATE TABLE ator_filme (
    fk_FILME_ID INT,
    fk_ATOR_ID INT
);

CREATE TABLE funcionario_pagamento (
    fk_PAGAMENTO_ID INT,
    fk_FUNCIONARIO_ID INT
);
```

```
CREATE TABLE cliente_pagamento (
    fk_PAGAMENTO_ID INT,
    fk_CLIENTE_ID INT
);

ALTER TABLE CLIENTE ADD CONSTRAINT FK_CLIENTE_2
    FOREIGN KEY (FK_LOJA_ID)
    REFERENCES LOJA (ID)
    ON DELETE RESTRICT;

ALTER TABLE FUNCIONARIO ADD CONSTRAINT FK_FUNCIONARIO_2
    FOREIGN KEY (FK_LOJA_ID)
    REFERENCES LOJA (ID)
    ON DELETE RESTRICT;

ALTER TABLE ALUGUEL ADD CONSTRAINT FK_ALUGUEL_2
    FOREIGN KEY (FK_CLIENTE_ID)
    REFERENCES CLIENTE (ID)
    ON DELETE RESTRICT;

ALTER TABLE ALUGUEL ADD CONSTRAINT FK_ALUGUEL_3
    FOREIGN KEY (FK_FUNCIONARIO_ID)
    REFERENCES FUNCIONARIO (ID)
    ON DELETE RESTRICT;

ALTER TABLE ITEM ADD CONSTRAINT FK_ITEM_2
    FOREIGN KEY (FK_ALUGUEL_ID)
    REFERENCES ALUGUEL (ID)
    ON DELETE RESTRICT;

ALTER TABLE ITEM ADD CONSTRAINT FK_ITEM_3
    FOREIGN KEY (FK_FILME_ID)
    REFERENCES FILME (ID)
    ON DELETE RESTRICT;

ALTER TABLE PAGAMENTO ADD CONSTRAINT FK_PAGAMENTO_2
    FOREIGN KEY (FK_ALUGUEL_ID)
    REFERENCES ALUGUEL (ID)
    ON DELETE RESTRICT;

ALTER TABLE PAGAMENTO ADD CONSTRAINT FK_PAGAMENTO_3
    FOREIGN KEY (FK_CLIENTE_ID)
    REFERENCES CLIENTE (ID)
    ON DELETE RESTRICT;

ALTER TABLE filme_categoria ADD CONSTRAINT FK_filme_categoria_1
    FOREIGN KEY (fk_FILME_ID)
```

```
    REFERENCES FILME (ID)
    ON DELETE RESTRICT;

ALTER TABLE filme_categoria ADD CONSTRAINT FK_filme_categoria_2
    FOREIGN KEY (fk_CATEGORIA_ID)
    REFERENCES CATEGORIA (ID)
    ON DELETE RESTRICT;

ALTER TABLE ator_filme ADD CONSTRAINT FK_ator_filme_1
    FOREIGN KEY (fk_FILME_ID)
    REFERENCES FILME (ID)
    ON DELETE RESTRICT;

ALTER TABLE ator_filme ADD CONSTRAINT FK_ator_filme_2
    FOREIGN KEY (fk_ATOR_ID)
    REFERENCES ATOR (ID)
    ON DELETE RESTRICT;

ALTER TABLE funcionario_pagamento ADD CONSTRAINT
FK_funcionario_pagamento_1
    FOREIGN KEY (fk_PAGAMENTO_ID)
    REFERENCES PAGAMENTO (ID)
    ON DELETE RESTRICT;

ALTER TABLE funcionario_pagamento ADD CONSTRAINT
FK_funcionario_pagamento_2
    FOREIGN KEY (fk_FUNCIONARIO_ID)
    REFERENCES FUNCIONARIO (ID)
    ON DELETE RESTRICT;

ALTER TABLE cliente_pagamento ADD CONSTRAINT
FK_cliente_pagamento_1
    FOREIGN KEY (fk_PAGAMENTO_ID)
    REFERENCES PAGAMENTO (ID)
    ON DELETE RESTRICT;

ALTER TABLE cliente_pagamento ADD CONSTRAINT
FK_cliente_pagamento_2
    FOREIGN KEY (fk_CLIENTE_ID)
    REFERENCES CLIENTE (ID)
    ON DELETE RESTRICT;
```



02)

a) Criar um UNIQUE INDEX para o atributo CPF na tabela Cliente.

CREATE UNIQUE INDEX idx_cpf_unique ON Cliente (cpf);

/*Este comando cria um índice único chamado idx_cpf_unique na coluna cpf da tabela cliente, garantindo que nenhum CPF seja repetido na tabela. /*

b) Criar as visões (VIEW) a seguir:

- Listagem dos atores e para cada ator uma descrição contendo os filmes nos quais ele atuou.

/*Esta view combina dados das tabelas ATOR e FILME para mostrar uma lista de atores e os filmes nos quais eles atuaram.

```
CREATE VIEW ListaAtoresFilmes AS
SELECT
   A.ID AS AtorID,
   A.nome AS AtorNome,
   F.titulo AS FilmeTitulo,
   F.descricao AS FilmeDescricao
FROM
   ATOR A
JOIN
   FILME F ON A.ID = F.ID;
```

Para selecionar dados da view:/*

```
SELECT * FROM ListaAtoresFilmes;
```

/*Isso retornará uma lista de atores com os títulos e descrições dos filmes nos quais atuaram./*

• Listar os filmes com suas categorias e atores.

/*Criando uma tabela associativa FILME_ATOR_CATEGORIA para relacionar filmes, atores e categorias. Assim, conseguimos criar uma view que lista os filmes com suas categorias e atores./*

```
CREATE TABLE FILME_ATOR_CATEGORIA (
   FILME_ID INT,
   ATOR_ID INT,
   CATEGORIA_ID INT,
   PRIMARY KEY (FILME_ID, ATOR_ID, CATEGORIA_ID),
   FOREIGN KEY (FILME_ID) REFERENCES FILME(ID),
   FOREIGN KEY (ATOR_ID) REFERENCES ATOR(ID),
   FOREIGN KEY (CATEGORIA_ID) REFERENCES
CATEGORIA(ID)
);
```

/*Esses comandos inserem dados na tabela FILME_ATOR_CATEGORIA e estabelece relações entre filmes, atores e categorias. Agora, podemos criar uma view que lista os filmes com suas categorias e atores:/*

```
INSERT INTO FILME_ATOR_CATEGORIA (FILME_ID, ATOR_ID, CATEGORIA_ID) VALUES
(1, 1, 1),
(1, 1, 3),
(2, 2, 4),
(3, 3, 1),
(3, 3, 3),
(4, 4, 4),
(5, 5, 2),
(5, 5, 5);
```

/* Essa view combina dados das tabelas FILME, ATOR, CATEGORIA e FILME_ATOR_CATEGORIA para mostrar uma lista de filmes com suas categorias e atores./*

```
CREATE VIEW ListaFilmesAtoresCategorias AS
SELECT
    F.ID AS FilmeID,
    F.titulo AS FilmeTitulo,
    A.nome AS AtorNome,
    C.nome AS CategoriaNome
FROM
    FILME F
JOIN
    FILME_ATOR_CATEGORIA FAC ON F.ID = FAC.FILME_ID
JOIN
    ATOR A ON FAC.ATOR_ID = A.ID
JOIN
    CATEGORIA C ON FAC.CATEGORIA_ID = C.ID;
```

/* Para selecionar dados da view:/*

```
SELECT * FROM ListaFilmesAtoresCategorias;
```

• Listar os 5 filmes mais alugados.

/* Essa view utiliza a tabela FILME e a tabela associativa ITEM para contar o número de aluguéis para cada filme. A lista é ordenada em ordem decrescente pelo número de aluguéis, e a cláusula LIMIT 5 garante que apenas os 5 filmes mais alugados sejam incluídos na view./*

```
CREATE VIEW Top5FilmesMaisAlugados AS
SELECT
    F.ID AS FilmeID,
    F.titulo AS FilmeTitulo,
```

```
    COUNT(I.FK_FILME_ID) AS NumeroAlugueis
FROM
    FILME F
JOIN
    ITEM I ON F.ID = I.FK_FILME_ID
GROUP BY
    F.ID, F.titulo
ORDER BY
    NumeroAlugueis DESC
LIMIT 5;
```

/* Essa consulta retornará os 5 filmes mais alugados, com seus IDs, títulos e o número total de aluguéis./*

```
SELECT * FROM Top5FilmesMaisAlugados;
```

• Listar os pagamentos em aberto e seus respectivos clientes.

/*Essa view utiliza as tabelas PAGAMENTO, ALUGUEL e CLIENTE para listar os pagamentos em aberto e seus respectivos clientes. O critério para um pagamento estar em aberto é que a data de pagamento (data_pagamento) seja nula./*

```
CREATE VIEW Pagacaloteirus AS
SELECT
    P.ID AS PagamentoID,
    P.valor AS ValorPagamento,
    P.data_pagamento AS DataPagamento,
    C.ID AS ClienteID,
    C.nome AS NomeCliente,
    C.email AS EmailCliente
FROM
    PAGAMENTO P
JOIN
    ALUGUEL A ON P.FK_ALUGUEL_ID = A.ID
JOIN
    CLIENTE C ON P.FK_CLIENTE_ID = C.ID
WHERE
    P.data_pagamento IS NULL;
```

/* Essa consulta retornará os pagamentos em aberto, incluindo os IDs dos pagamentos, valores, datas de pagamento e informações dos clientes associados./*

```
SELECT * FROM Pagacaloteirus;
```

• Listagem das lojas, com seu gerente.

/*Essa view utiliza as tabelas LOJA e FUNCIONARIO para listar as lojas com seus respectivos gerentes. A relação entre as tabelas é estabelecida pelo campo gerente na tabela LOJA, que é uma referência ao campo login na tabela FUNCIONARIO./*

```
CREATE VIEW ListagemChefinhoDasLojas AS
SELECT
    L.ID AS LojaID,
    L.gerente AS GerenteLogin,
    F.ID AS FuncionarioID,
    F.nome AS NomeGerente,
    F.email AS EmailGerente
FROM
    LOJA L
JOIN
    FUNCIONARIO F ON L.gerente = F.login;
```

/*Essa consulta retornará uma lista das lojas com seus IDs, gerentes de loja, IDs dos gerentes (funcionários) e informações adicionais sobre esses gerentes./*

```
SELECT * FROM ListagemChefinhoDasLojas;
```

• Listar os clientes das lojas em ordem alfabética por loja e cliente.

/*Esta view combina informações das tabelas LOJA e CLIENTE, mostrando o ID da loja, o gerente da loja, o ID do cliente, o nome do cliente e o e-mail do cliente. Os resultados são ordenados por ID da loja e nome do cliente./*

```
CREATE VIEW ListarBencasDePaciencia AS
SELECT
    L.ID AS LojaID,
    L.gerente AS Gerente,
    C.ID AS ClienteID,
    C.nome AS NomeCliente,
    C.email AS EmailCliente
FROM
    LOJA L
JOIN
    CLIENTE C ON L.ID = C.FK_LOJA_ID
ORDER BY
    L.ID, C.nome;
```

/* Ao executar essa consulta, você verá um conjunto de resultados que inclui todas as informações disponíveis na visão para cada cliente e loja./*

```
SELECT * FROM ListarBencasDePaciencia;
```

c) Acrescentar a tabela Estoque e os relacionamentos no modelo físico, seguindo o modelo abaixo:

```
CREATE TABLE ESTOQUE (
    ID INT PRIMARY KEY,
    quantidade_disponivel INT,
    FK_FILME_ID INT,
    FK_LOJA_ID INT,
    FOREIGN KEY (FK_FILME_ID) REFERENCES FILME (ID),
    FOREIGN KEY (FK_LOJA_ID) REFERENCES LOJA (ID)
);

ALTER TABLE ITEM
ADD COLUMN FK_ESTOQUE_ID INT,
ADD CONSTRAINT FK_ITEM_4
    FOREIGN KEY (FK_ESTOQUE_ID)
    REFERENCES ESTOQUE (ID)
    ON DELETE RESTRICT;
```

/*Neste modelo atualizado, a tabela ESTOQUE foi adicionada, incluindo a quantidade disponível, bem como chaves estrangeiras para as tabelas FILME e LOJA. A tabela ITEM foi alterada para incluir a coluna FK_ESTOQUE_ID e uma chave estrangeira correspondente a essa coluna foi adicionada./*

d) Criar a seguinte Function:

• Controle de Inventário – Retorna 1 quando o item está no estoque ou 0 quando o item está emprestado. Obs: Um item consta no estoque se não existir nenhum registro deste item alugado ou se os registros que existirem estiverem com a data de devolução preenchida.

```
DELIMITER $
CREATE FUNCTION ControleInventario(itemID INT) RETURNS INT
BEGIN
    DECLARE emprestado INT;

  /*Verificar se o item está emprestado
    SELECT COUNT(*)
    INTO emprestado
    FROM ALUGUEL A
    JOIN ITEM I ON A.ID = I.FK_ALUGUEL_ID
```

```
    WHERE I.FK_ESTOQUE_ID = itemID AND A.data_devolucao IS
NULL;

/* Retorna 1 se não estiver emprestado, 0 se estiver emprestado
    RETURN IF(emprestado = 0, 1, 0);
END $
DELIMITER ;
```

/* A função ControleInventario recebe o ID do item como parâmetro e utiliza uma consulta para verificar se existem registros de aluguel não devolvidos para esse item. Se não houver nenhum registro não devolvido, a função retorna 1, indicando que o item está no estoque; caso contrário, retorna 0, indicando que o item está emprestado.

Para usar a função, você pode fazer chamadas como a seguinte:/*

```
SELECT ControleInventario(1);
```

e) Criar as seguintes Stored Procedures:

• **Ator –** Listar o nome dos atores limitando ao número de registros a serem retornados.

```
DELIMITER $
CREATE PROCEDURE ListarAtoresComLimite(IN limite INT)
BEGIN
    SELECT nome
    FROM ATOR
    LIMIT limite;
END $
DELIMITER ;
```

/*Nesta Stored Procedure chamada ListarAtoresComLimite, ela recebe um parâmetro limite que determina o número máximo de registros a serem retornados. Dentro do procedimento, realizamos uma simples consulta SELECT na tabela ATOR e utilizamos a cláusula LIMIT para limitar o número de resultados.
Para chamar a Stored Procedure e obter a lista de atores com um limite específico, você pode executar o seguinte comando:/*

```
CALL ListarAtoresComLimite(5);
```

• **Filmes em Estoque –** Retornar a quantidade de filmes no estoque, informando o id do filme e o id da loja.

```
/*Criar a Stored Procedure para retornar a quantidade de filmes no estoque/*
DELIMITER $
CREATE PROCEDURE FilmesEmEstoque(IN filmeID INT, IN lojaID INT, OUT quantidade INT)
BEGIN
/*Contar a quantidade de filmes no estoque para o filme e a loja especificados/*
   SELECT COALESCE(SUM(E.quantidade_disponivel), 0)
   INTO quantidade
   FROM ESTOQUE E
   WHERE E.FK_FILME_ID = filmeID AND E.FK_LOJA_ID = lojaID;
END $
DELIMITER ;
```

```
/*Para chamar a Stored Procedure e obter a quantidade de filmes no estoque para um filme e loja específicos, você pode executar o seguinte comando:/*
CALL FilmesEmEstoque(4, 1, @quantidade);
SELECT @quantidade AS QuantidadeFilmesEmEstoque;
```

```
/*Neste exemplo , a chamada está pedindo para a Stored Procedure calcular a quantidade de filmes no estoque para o filme com ID 4 e na loja com ID 2, e armazenar o resultado na variável @quantidade./*
```

• **Filmes alugados –** Retorna a quantidade de filmes alugados, informados o id do filme e o id da loja. Um filme alugado tem a data da devolução NULL.

```
DELIMITER $
CREATE PROCEDURE FilmesAlugados(IN filmeID INT, IN lojaID INT, OUT quantidade INT)
BEGIN
/*Contar a quantidade de filmes alugados para o filme e a loja especificados/*
   SELECT COUNT(*)
   INTO quantidade
   FROM ALUGUEL A
   JOIN ITEM I ON A.ID = I.FK_ALUGUEL_ID
   JOIN CLIENTE C ON A.FK_CLIENTE_ID = C.ID
   JOIN FUNCIONARIO F ON A.FK_FUNCIONARIO_ID = F.ID
   WHERE I.FK_FILME_ID = filmeID AND (C.FK_LOJA_ID = lojaID OR F.FK_LOJA_ID = lojaID) AND A.data_devolucao IS NULL;
END $
```

```
DELIMITER ;
```

/*Neste código, foi adicionado JOIN às tabelas CLIENTE e FUNCIONARIO para buscar informações da loja associada a cada aluguel. A condição C.FK_LOJA_ID = lojaID OR F.FK_LOJA_ID = lojaID permite que a loja seja identificada através do cliente ou do funcionário associado ao aluguel./*

/*Variável para armazenar a quantidade de filmes alugados/*

```
SET @quantidade_filmes_alugados = 0;
```

/*Chamada da Stored Procedure/*

```
CALL FilmesAlugados(8, 8, @quantidade_filmes_alugados);
```

/*Exibição do resultado/*

```
SELECT @quantidade_filmes_alugados AS QuantidadeFilmesAlugados;
```

f) Alterar a base de dados:

• Alterar a tabela cliente e acrescentar um campo com o nome “criado_em” do tipo “datetime”.

/*Esse comando adicionará uma nova coluna chamada "criado_em" à tabela CLIENTE com o tipo de dado "datetime". Esta operação é irreversível e pode afetar a estrutura existente da tabela./*

```
ALTER TABLE CLIENTE
ADD COLUMN criado_em DATETIME;
```

g) Criar uma trigger “cliente_BEFORE_UPDATE”

• A trigger deve atualizar a data de criação de cada registro automaticamente.

```
DELIMITER $
CREATE TRIGGER cliente_BEFORE_UPDATE
BEFORE UPDATE ON CLIENTE
FOR EACH ROW
SET NEW.criado_em = NOW();
$
DELIMITER ;
```

/*Essa trigger, denominada cliente_BEFORE_UPDATE, será acionada antes de cada operação de UPDATE na tabela CLIENTE. Ela atualiza a coluna "criado_em" para o valor atual da data e hora usando a função NOW().
Essa trigger adicionará automaticamente a data de criação atual sempre que um registro da tabela CLIENTE for atualizado./*

h) Criar a tabela “log”.
• A tabela deve ter dois campos “reg” e “msg”, ambos do tipo varchar(255).

```
CREATE TABLE log (
   reg VARCHAR(255),
   msg VARCHAR(255)
);
```

/*Esse comando cria uma tabela chamada "log" com dois campos: "reg" e "msg", ambos do tipo varchar(255)./*

i) Criar uma trigger “cliente_AFTER_UPDATE”
• A trigger deve inserir um registro na tabela “log” contendo as informações de cada nova inserção na tabela cliente.

```
DELIMITER $
CREATE TRIGGER cliente_AFTER_UPDATE
AFTER UPDATE ON CLIENTE
FOR EACH ROW
BEGIN
   -- Inserir informações na tabela log após cada atualização
   INSERT INTO log (reg, msg)
   VALUES ('CLIENTE', CONCAT('Registro ID: ', NEW.ID, ' atualizado em ', NOW()));
END;
$
DELIMITER ;
```

/*Esta trigger, cliente_AFTER_UPDATE, será acionada após cada operação de UPDATE na tabela CLIENTE. Ela insere um registro na tabela log contendo informações como o tipo de registro ("CLIENTE"), o ID do registro atualizado e a data/hora da atualização./*

J) Criar um novo USUÁRIO.

```
CREATE USER 'toquerendodormir'@'localhost' IDENTIFIED BY '123456';
```

/*Este comando cria um usuário chamado "toquerendodormir" que pode se conectar apenas a partir do localhost (máquina onde o MySQL está sendo executado)./*

K) Atribuir os seguintes privilégios:
• Leitura, atualização, exclusão e inserção em todas as tabelas.

GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO 'toquerendodormir'@'localhost' WITH GRANT OPTION;

/*Este comando concede os privilégios de seleção (leitura), inserção, atualização e exclusão em todas as tabelas (*.*) para o usuário "toquerendodormir". A opção WITH GRANT OPTION permite que o usuário conceda esses privilégios a outros usuários./*

L) Revogar os seguintes privilégios:
• Atualização, exclusão e inserção da tabela Pagamento.

/*Se deseja revogar os privilégios de atualização, exclusão e inserção na tabela "Pagamento" do usuário "toquerendodormir", você pode usar o seguinte comando:/*
REVOKE UPDATE, DELETE, INSERT ON niltaomeaprova.PAGAMENTO FROM 'toquerendodormir'@'localhost';

MAIKON FABRÍCIO GINO **RA: 2901392313007**