BD fichas

Ficha 2

Untitled Attachment

1. Employee

2. σ sex='Feminino' (Employee)

3. π fName, lName, address ((Department) ⋈ Department.mgrEmpNo=Employee.empNo (Employee))

4. π fName, lName, address (σ deptName='IT' (Department) ⋈ Employee)

5. π fName, lName (Employee ⋈ Employee.empNo=WorksOn.empNo (σ projName='SCCS'(Project) ⋈ WorksOn))

6. σ DOB<date('1955-01-01')(Employee)

7. γ mgrEmpNo;count(empNo)→contagem (π mgrEmpNo, Department.deptNo (σ fName='James' ∧ lName='Adams' (Employee ⋈ Employee.empNo=Department.mgrEmpNo (Department))) ⋈ deptNo=Employee.deptNo (Employee))

8. Employee ⋈ (γ empNo;sum(hoursWorked)→horasTotais (π empNo, hoursWorked (WorksOn)))

9. π projNo, projName, totalEmp (σ totalEmp > 2 ((γ projNo;count(empNo) → totalEmp (π empNo, projNo (WorksOn))) ⋈ Project))

10. ρ Departamento←d.deptNo π d.deptNo, totalEmp σ totalEmp > 10 γ d.deptNo; COUNT(e.empNo)→totalEmp (ρ e Employee ⋈ e.deptNo = d.deptNo ρ d Department)

```
group: Empresa

Employee = {
empNo:number, fName:string, lName:string, address:string, DOB:date,
sex:string, position:string, deptNo:number
```

```
123, Joao, Pedro, Felgueiras, 1980-01-20, Masculino, Programador, 1
741, Ana, Barbosa, Valongo, 1982-04-10, Feminino, Programador, 1
509, Adolfo, Marques, Agueda, 1981-07-11, Masculino, Engenheiro, 1
357, Julia, Santos, Viseu, 1980-08-25, Feminino, Programador, 2
219, Joana, Faria, Lisboa, 1980-05-25, Feminino, Engenheiro, 3
951, Tomas, Pereira, 'Viana do Castelo', 1984-11-12, Masculino,
Programador, 4
198, Pedro, Miguel, Porto, 1978-10-06, Masculino, Gestor, 2
212, Teresa, Lopes, Lisboa, 1981-01-25, Feminino, Engenheiro, 4
801, Carlos , Alberto,Braga, 1981-08-25, Masculino, Gestor, 3
658, James, Adams, Londres, 1977-06-05, Masculino, Gestor, 1
777, James, Adams, Porto, 1977-06-05, Masculino, Gestor, 4
}

Department = {
deptNo:number, deptName:string, mgrEmpNo:number
1, IT, 658
2, Administracao, 741
3, 'Recursos Humanos', 801
4, Contabilidade, 777
}

Project = {
projNo:number, projName:string, deptNo:number
1, SCCS, 1
2, ERP, 2
3, Prim, 3
4, TopX, 1
5, CRM, 4
}

WorksOn = {
empNo:number, projNo:number, dateWorked:date, hoursWorked:number
123, 1, 2000-02-05, 10
123, 2, 2000-09-10, 2
741, 1, 2000-03-02, 5
509, 1, 2000-04-19, 7
357, 2, 2001-06-05, 6
219, 3, 1999-06-05, 7
951, 5, 2002-06-05, 8
357, 1, 2001-12-20, 8
198, 2, 2003-06-05, 5
509, 2, 2001-05-21, 6
212, 5, 2002-06-05, 5
```

```
357, 4, 2003-02-20, 9
509, 3, 2002-10-19, 12
801, 4, 1999-06-05, 11
123, 3, 2001-09-13, 5
658, 2, 2000-06-05, 11
777, 5, 2002-07-03, 12
}
```

Ficha 3

Untitled Attachment

1. SELECT * FROM Employee order by lName,fName;
2. SELECT * FROM Employee where sex='Feminino';
3. SELECT e.fName,e.lName,e.address FROM Employee as e JOIN Department as d on e.empNo = d.mgrEmpNo;
4. SELECT e.fName,e.lName, e.address FROM Employee as e INNER JOIN Department as d on e.deptNo = d.deptNo where d.deptName='IT';
5. SELECT e.fName,e.lName FROM Employee as e JOIN WorksOn as w on e.empNo = w.empNo JOIN Project as p on w.projNo=p.projNo where p.projName='SCCS'
6. SELECT * FROM Employee where DOB < date('1990-01-01') order by lName;
7. SELECT d.mgrEmpNo, COUNT(e1.empNo) as contagem FROM Employee as e JOIN Department as d on e.empNo=d.mgrEmpNo JOIN Employee as e1 on d.deptNo=e1.deptNo where e.fName='James' and e.lName='Adams' group by d.mgrEmpNo;
8. SELECT e.empNo, e.fName, e.lName, e.deptNo, SUM(w.hoursWorked) as horasTotais FROM Employee as e JOIN WorksOn as w on e.empNo=w.empNo group by e.empNo,e.deptNo, e.fName, e.lName order by e.deptNo, e.lName;
9. SELECT p.projNo, p.projName, COUNT(e.empNo) as totalEmp FROM Employee as e JOIN WorksOn as w on e.empNo=w.empNo JOIN Project as p on w.projNo=p.projNo group by p.projNo, p.projName having totalEmp > 2;
10. SELECT d.deptNo as Departamento, COUNT(e.empNo) as totalEmp FROM Employee as e JOIN Department as d on e.deptNo=d.deptNo group by d.deptNo having totalEmp > 10;

Ficha 4

Untitled Attachment

1. SELECT * FROM Hotel;
2. SELECT * FROM Hotel WHERE city='London';

4. SELECT * FROM Room WHERE (type='double' OR type='Family') AND price<40 ORDER BY price ;
5. SELECT * FROM Booking WHERE dateTo IS NULL ;
6. SELECT COUNT(*) as QtdHotels FROM Hotel;
7. SELECT AVG(price) as media FROM Room;
8. SELECT SUM(price) as total FROM Room WHERE type='double';
9. SELECT DISTINCT COUNT(guestNo) as total FROM Booking WHERE dateFrom Like '%-08-%' OR dateTo Like '%-08-%'; ou SELECT DISTINCT COUNT(guestNo) as total FROM Booking WHERE month(dateFrom)<=8 OR month(dateTo)>=8;
10. SELECT price, type FROM Room WHERE hotelNo IN (SELECT hotelNo FROM Hotel WHERE hotelName='Grosvenor Hotel');
11. SELECT * FROM Guest WHERE guestNo IN (SELECT guestNo FROM Booking WHERE '2020-04-03' BETWEEN dateFrom AND dateTo AND hotelNo IN (SELECT hotelNo FROM Hotel WHERE hotelName='Grosvenor Hotel'));
12. SELECT r.*, g.guestName FROM Room r LEFT JOIN Booking b ON r.roomNo = b.roomNo AND b.hotelNo = r.hotelNo AND '2020-04-03' BETWEEN dateFrom AND dateTo LEFT JOIN Guest g ON b.guestNo = g.guestNo WHERE r.hotelNo IN (SELECT hotelNo FROM Hotel WHERE hotelName='Grosvenor Hotel');
13. SELECT SUM(price) as Total FROM Booking b JOIN Room r on b.roomNo=r.roomNo AND b.hotelNo=r.hotelNo WHERE '2020-04-03' BETWEEN dateFrom AND dateTo AND b.hotelNo IN (SELECT hotelNo FROM Hotel WHERE hotelName='Grosvenor Hotel');
14. SELECT r.roomNo, r.hotelNo FROM Room r LEFT JOIN Booking b ON r.roomNo = b.roomNo AND b.hotelNo = r.hotelNo AND '2020-04-03' BETWEEN dateFrom AND dateTo WHERE r.hotelNo IN (SELECT hotelNo FROM Hotel WHERE hotelName='Grosvenor Hotel') AND b.guestNo IS NULL;
15. SELECT SUM(price) FROM Room r LEFT JOIN Booking b ON r.roomNo = b.roomNo AND b.hotelNo = r.hotelNo AND '2020-04-03' BETWEEN dateFrom AND dateTo WHERE r.hotelNo IN (SELECT hotelNo FROM Hotel WHERE hotelName='Grosvenor Hotel') AND b.guestNo IS NULL;
16. SELECT h.hotelNo, COUNT(r.roomNo) as TotalQuartosFROM Room r LEFT JOIN Hotel h ON h.hotelNo = r.hotelNo GROUP BY h.hotelNo;
17. SELECT h.hotelNo, h.hotelName, COUNT(r.roomNo) as TotalQuartos FROM Room r LEFT JOIN Hotel h ON h.hotelNo = r.hotelNo WHERE h.city = 'London' GROUP BY h.hotelNo, h.hotelName;
18. SELECT AVG(cnt_bookings) AS MediaAgosto FROM (SELECT b.hotelNo, COUNT(*) AS cnt_bookings FROM Hotel h INNER JOIN Booking b ON h.hotelNo = b.hotelN WHERE month(dateFrom)=8 OR month(dateTo)=8 GROUP BY b.hotelNo AS hotel_bookings;

19. SELECT TOP 1 r.typeFROM Hotel h INNER JOIN Booking b ON b.hotelNo = h.hotelNo INNER JOIN Room r ON r.roomNo = b.roomNo AND b.hotelNo = r.hotelNo WHERE h.city = 'London'GROUP BY r.type ORDER BY COUNT(*) DESC;
20. SELECT r.hotelNo, SUM(price) as prejuizoQuartos FROM Room r INNER JOIN Hotel h ON r.hotelNo = h.hotelNo LEFT JOIN Booking b ON r.roomNo = b.roomNo AND b.hotelNo = r.hotelNo AND '2020-04-03' BETWEEN dateFrom AND dateTo WHERE b.guestNo IS NULL GROUP BY r.hotelNo ;
21. INSERT INTO Hotel VALUES(101,'Hotel Maria do Carmo', 'Mindelo'); INSERT INTO Room VALUES(1, 101,'single', 100); INSERT INTO Guest VALUES(1001,'David Santos', 'Felgueiras');INSERT INTO Guest VALUES(1001,'David Santos', 'Felgueiras'); INSERT INTO Booking VALUES(101,1001,'2024-04-13', '2024-04-30', 1);
22. UPDATE Room SET price = price * 1.05;

Exercicio 1

```
CREATE TABLE EscolaCurso (
codEscola INTEGER CHECK (codEscola BETWEEN 0 AND 99), -- Código da
Escola
codCurso INTEGER CHECK (codCurso BETWEEN 0 AND 9999), -- Código do
Curso
Data DATE, -- Data de início do curso na escola
NotaUltimoColocado DECIMAL(4, 2) CHECK (NotaUltimoColocado BETWEEN 9.50
AND 20.00), -- Nota do último colocado no curso
Obs VARCHAR(1000), -- Campo de observações
PRIMARY KEY (codEscola, codCurso),
FOREIGN KEY (codEscola) REFERENCES Escola(codEscola),
FOREIGN KEY (codCurso) REFERENCES Curso(codCurso)
);


--------------
-- Tabela Escola
CREATE TABLE Escola (
codEscola INT NOT NULL PRIMARY KEY IDENTITY(1,1),
Nome VARCHAR(60) NOT NULL,
Sigla VARCHAR(10) NOT NULL,
Obs VARCHAR(1000),
CONSTRAINT Sigla_UNIQUE UNIQUE (Sigla)
);
-- Tabela Curso
CREATE TABLE Curso (
codCurso INT NOT NULL PRIMARY KEY IDENTITY(1,1),
```

```
Nome VARCHAR(100) NOT NULL,
Obs VARCHAR(1000),
CONSTRAINT check_codCurso_range CHECK (codCurso < 1500 OR codCurso >
2300)
);
-- Tabela EscolaCurso
CREATE TABLE EscolaCurso (
codEscola INT,
codCurso INT,
Data DATE NOT NULL,
NotaUltimoColocado DECIMAL(4,2),
Obs VARCHAR(1000),
PRIMARY KEY (codEscola, codCurso),
FOREIGN KEY (codEscola) REFERENCES Escola(codEscola),
FOREIGN KEY (codCurso) REFERENCES Curso(codCurso)
);
-- Tabela Aluno
CREATE TABLE Aluno (
codAluno VARCHAR(10) NOT NULL PRIMARY KEY,
codEscola INT,
codCurso INT,
Nome VARCHAR(150) NOT NULL,
Sexo CHAR(1) NOT NULL,
Idade TINYINT NOT NULL,
NotaColocado DECIMAL(4,2) NOT NULL,
FOREIGN KEY (codEscola) REFERENCES Escola(codEscola),
FOREIGN KEY (codCurso) REFERENCES Curso(codCurso) ON DELETE NO ACTION,
CONSTRAINT check_sexo CHECK (Sexo IN ('M', 'F'))
);
-- Trigger para garantir que um curso não tenha mais de 100 alunos
CREATE TRIGGER CheckMaxStudents
ON Aluno
AFTER INSERT
AS
BEGIN
DECLARE @num_students INT;
SELECT @num_students = COUNT(*) FROM Aluno WHERE codCurso = (SELECT
codCurso FROM inserted);
IF @num_students >= 100
BEGIN
RAISERROR ('Um curso não pode ter mais de 100 alunos.', 16, 1);
ROLLBACK TRANSACTION;
END;
END;
```

```
CREATE TABLE Hospital (
codH CHAR(6) PRIMARY KEY,
Designacao VARCHAR(100) NOT NULL
);

CREATE TABLE Quarto (
codHospital CHAR(6) NOT NULL,
codQuarto VARCHAR(10) NOT NULL,
Designacao VARCHAR(100) NOT NULL,
Especialidade CHAR(2) NOT NULL CHECK (Especialidade IN ('N', 'A', 'B',
'AB', 'O')),
PRIMARY KEY (codHospital, codQuarto),
FOREIGN KEY (codHospital) REFERENCES Hospital(codH),
CONSTRAINT check_codQuarto_length CHECK (LEN(codQuarto) BETWEEN 4 AND
10)
);


CREATE TABLE Paciente (
codP INT PRIMARY KEY,
Designacao VARCHAR(200) NOT NULL,
nUtente INT NOT NULL,
sexo CHAR(1) CHECK (sexo IN ('M', 'F')),
CONSTRAINT check_codP_length CHECK (codP >= 0 AND codP <= 99999),
CONSTRAINT check_nUtente_range CHECK (nUtente BETWEEN 1000000000 AND
999999999)
);

-- Adicionando restrições adicionais
-- a) Restrição para garantir que um quarto de hospital tenha
capacidade para apenas um paciente num dado período
ALTER TABLE Internamento
ADD CONSTRAINT unique_quarto_capacidade UNIQUE (codHospital, codQuarto,
dataSaidaReal);
-- b) Restrição para garantir que dataPrevisaoSaida e dataSaidaReal não
sejam inferiores a DataEntrada
ALTER TABLE Internamento
ADD CONSTRAINT check_datas CHECK (dataPrevisaoSaida >= DataEntrada AND
dataSaidaReal >= DataEntrada);
-- c) Restrições de NOT NULL para todos os atributos exceto Obs e
dataSaidaReal da relação Internamento
```

```
ALTER TABLE Internamento
ALTER COLUMN codHospital CHAR(6) NOT NULL,
ALTER COLUMN codQuarto VARCHAR(10) NOT NULL,
ALTER COLUMN codPaciente INT NOT NULL,
ALTER COLUMN DataEntrada DATE NOT NULL,
ALTER COLUMN Sintomas VARCHAR(500) NOT NULL,
ALTER COLUMN dataPrevisaoSaida DATE NOT NULL;
-- d) Restrição ON DELETE CASCADE para eliminar automaticamente quartos
e internamentos associados a um hospital excluído
ALTER TABLE Quarto
ADD CONSTRAINT fk_hospital_quarto FOREIGN KEY (codHospital) REFERENCES
Hospital(codH) ON DELETE CASCADE;
ALTER TABLE Internamento
ADD CONSTRAINT fk_hospital_internamento FOREIGN KEY (codHospital)
REFERENCES Hospital(codH) ON DELETE CASCADE;
```

Ficha 7

Untitled Attachment

Ficha 8

Untitled Attachment

1. a)

```
-- Procedimento para inserir um novo produto
CREATE PROCEDURE InsertProduct
@ProductId NVARCHAR(15),
@ProductName NVARCHAR(150),
@FamilyId NVARCHAR(10),
@Ean13Code NVARCHAR(13),
@Obs NVARCHAR(MAX)
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
BEGIN TRANSACTION;
INSERT INTO Product (ProductId, ProductName, FamilyId, Ean13Code, Obs,
InsertUserId, InsertDateTime)
VALUES (@ProductId, @ProductName, @FamilyId, @Ean13Code, @Obs,
ORIGINAL_LOGIN(), GETDATE());
COMMIT TRANSACTION;
END TRY
BEGIN CATCH
IF @@TRANCOUNT > 0
ROLLBACK TRANSACTION;
THROW;
```

```
END CATCH;
END;
GO
-- Procedimento para alterar um produto existente
CREATE PROCEDURE UpdateProduct
@ProductId NVARCHAR(15),
@ProductName NVARCHAR(150),
@FamilyId NVARCHAR(10),
@Ean13Code NVARCHAR(13),
@Obs NVARCHAR(MAX)
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
BEGIN TRANSACTION;
UPDATE Product
SET ProductName = @ProductName,
FamilyId = @FamilyId,
Ean13Code = @Ean13Code,
Obs = @Obs,
LastModifiedDateTime = GETDATE(),
LastModifiedUserId = ORIGINAL_LOGIN()
WHERE ProductId = @ProductId;
COMMIT TRANSACTION;
END TRY
BEGIN CATCH
IF @@TRANCOUNT > 0
ROLLBACK TRANSACTION;
THROW;
END CATCH;
END;
GO
-- Procedimento para remover um produto
CREATE PROCEDURE DeleteProduct
@ProductId NVARCHAR(15)
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
BEGIN TRANSACTION;
DELETE FROM Product
WHERE ProductId = @ProductId;
COMMIT TRANSACTION;
END TRY
```

```
BEGIN CATCH
IF @@TRANCOUNT > 0
ROLLBACK TRANSACTION;
THROW;
END CATCH;
END;
GO
```

B

```
CREATE TABLE ErrorLog (
ErrorLogId INT IDENTITY(1,1) PRIMARY KEY,
ErrorMessage NVARCHAR(MAX),
ErrorProcedureName NVARCHAR(255),
ErrorDateTime DATETIME
);
```

Parte do catch nas procedures criadas altera para:

```
BEGIN CATCH
IF @@TRANCOUNT > 0
ROLLBACK TRANSACTION;
-- Registrar o erro na tabela de log de erros
INSERT INTO ErrorLog (ErrorMessage, ErrorProcedureName, ErrorDateTime)
VALUES (ERROR_MESSAGE(), 'InsertProduct', GETDATE());
THROW;
END CATCH;
```

c.

```
USE [warehouseDB]
GO
/****** Object: Table [dbo].[StocksDetails] Script Date: 26/04/2024
11:33:35 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[TempStocksDetails](
[SessionID] [INT] NOT NULL,
[DocumentType] [char](1) NOT NULL,
[DocumentNumber] [bigint] NOT NULL,
[DocumentNumberLine] [smallint] NOT NULL,
[WatehouseId] [nvarchar](10) NOT NULL,
[ProductId] [nvarchar](15) NOT NULL,
[Quantity] [decimal](18, 5) NOT NULL,
[UnitValue] [decimal](18, 4) NOT NULL,
[TotalLineValue] AS ([Quantity]*[UnitValue]) PERSISTED,
[InsertUserId] [nvarchar](30) NOT NULL,
```

```
[InsertDateTime] [datetime] NOT NULL,
[LastModifiedUserId] [nvarchar](30) NULL,
[LastModifiedDateTime] [datetime] NULL,
CONSTRAINT [PK_TempStocksDetails] PRIMARY KEY CLUSTERED
(
[SessionID] ASC,
[DocumentType] ASC,
[DocumentNumber] ASC,
[DocumentNumberLine] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[TempStocksDetails] ADD CONSTRAINT
[DF_TempStocksDetails_InsertUserId] DEFAULT (original_login()) FOR
[InsertUserId]
GO
ALTER TABLE [dbo].[TempStocksDetails] ADD CONSTRAINT
[DF_TempStocksDetails_InsertDateTime] DEFAULT (getdate()) FOR
[InsertDateTime]
GO
ALTER TABLE [dbo].[TempStocksDetails] WITH CHECK ADD CONSTRAINT
[FK_TempStocksDetails_Product] FOREIGN KEY([ProductId])
REFERENCES [dbo].[Product] ([ProductId])
GO
ALTER TABLE [dbo].[TempStocksDetails] CHECK CONSTRAINT
[FK_TempStocksDetails_Product]
GO
ALTER TABLE [dbo].[TempStocksDetails] WITH CHECK ADD CONSTRAINT
[FK_TempStocksDetails_StocksHeader] FOREIGN KEY([DocumentType],
[DocumentNumber])
REFERENCES [dbo].[StocksHeader] ([DocumentType], [DocumentNumber])
GO
ALTER TABLE [dbo].[TempStocksDetails] CHECK CONSTRAINT
[FK_TempStocksDetails_StocksHeader]
GO
ALTER TABLE [dbo].[TempStocksDetails] WITH CHECK ADD CONSTRAINT
[FK_TempStocksDetails_Warehouse] FOREIGN KEY([WatehouseId])
REFERENCES [dbo].[Warehouse] ([WarehouseId])
GO
ALTER TABLE [dbo].[TempStocksDetails] CHECK CONSTRAINT
[FK_TempStocksDetails_Warehouse]
GO
```

```
ALTER TABLE [dbo].[TempStocksDetails] WITH CHECK ADD CONSTRAINT
[CK_maiorque0] CHECK (([DocumentNumberLine]>=(0) AND [UnitValue]>=(0)))
GO
ALTER TABLE [dbo].[TempStocksDetails] CHECK CONSTRAINT [CK_maiorque0]
GO
```

Procedimento para Inserir Dados na Tabela Temporária

```
CREATE PROCEDURE InsertTempStockDetail
@SessionID INT,
@DocumentType CHAR(1),
@DocumentNumber BIGINT,
@DocumentNumberLine SMALLINT,
@WarehouseId NVARCHAR(10),
@ProductId NVARCHAR(15),
@Quantity DECIMAL(18, 5),
@UnitValue DECIMAL(18, 4),
@InsertUserId NVARCHAR(30) = NULL,
@InsertDateTime DATETIME = NULL,
@LastModifiedUserId NVARCHAR(30) = NULL,
@LastModifiedDateTime DATETIME = NULL
AS
BEGIN
SET NOCOUNT ON;
INSERT INTO dbo.TempStocksDetails
(
SessionID,
DocumentType,
DocumentNumber,
DocumentNumberLine,
WarehouseId,
ProductId,
Quantity,
UnitValue,
InsertUserId,
InsertDateTime,
LastModifiedUserId,
LastModifiedDateTime
)
VALUES
(
@SessionID,
@DocumentType,
@DocumentNumber,
@DocumentNumberLine,
@WarehouseId,
```

```
@ProductId,
@Quantity,
@UnitValue,
COALESCE(@InsertUserId, ORIGINAL_LOGIN()),
COALESCE(@InsertDateTime, GETDATE()),
@LastModifiedUserId,
@LastModifiedDateTime
);
END;
```

Procedimentos para Inserir um Novo Movimento de Stock

```
CREATE PROCEDURE InsertStocksHeader
@DocumentType CHAR(1),
@DocumentNumber BIGINT,
@DocumentDate DATE,
@InsertUserId NVARCHAR(30) = NULL,
@InsertDateTime DATETIME = NULL,
@LastModifiedUserId NVARCHAR(30) = NULL,
@LastModifiedDateTime DATETIME = NULL
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
BEGIN TRANSACTION;
INSERT INTO dbo.StocksHeader
(
DocumentType,
DocumentNumber,
DocumentDate,
InsertUserId,
InsertDateTime,
LastModifiedUserId,
LastModifiedDateTime
)
VALUES
(
@DocumentType,
@DocumentNumber,
COALESCE(@DocumentDate, GETDATE()),
COALESCE(@InsertUserId, ORIGINAL_LOGIN()),
COALESCE(@InsertDateTime, GETDATE()),
@LastModifiedUserId,
@LastModifiedDateTime
);
COMMIT TRANSACTION;
```

```
END TRY
BEGIN CATCH
ROLLBACK TRANSACTION;
THROW;
END CATCH
END;
```

por terminar

d.

```
CREATE TRIGGER trg_InsertProductStock
ON dbo.Product
AFTER INSERT
AS
BEGIN
SET NOCOUNT ON;
-- Insere registros na tabela Stock para cada armazém existente com
stock 0
INSERT INTO dbo.Stock (ProductId, WarehouseId, StockAvailable,
InsertUserId, InsertDateTime)
SELECT i.ProductId, w.WarehouseId, 0, ORIGINAL_LOGIN(), GETDATE()
FROM inserted i
CROSS JOIN dbo.Warehouse w;
END;
GO
```

e.

```
CREATE TRIGGER trg_InsertWarehouseStock
ON dbo.Warehouse
AFTER INSERT
AS
BEGIN
SET NOCOUNT ON;
INSERT INTO dbo.Stock (ProductId, WarehouseId, StockAvailable,
InsertUserId, InsertDateTime)
SELECT p.ProductId, i.WarehouseId, 0, ORIGINAL_LOGIN(), GETDATE()
FROM inserted i
CROSS JOIN dbo.Product p;
END;
GO
```

f.

```
CREATE TRIGGER trg_PreventDeleteProductWithStock
ON dbo.Product
INSTEAD OF DELETE
```

```
AS
BEGIN
SET NOCOUNT ON;
-- Verificar se algum dos produtos a serem deletados tem estoque
disponível
IF EXISTS (
SELECT 1
FROM deleted d
JOIN dbo.Stock s ON d.ProductId = s.ProductId
WHERE s.StockAvailable > 0
)
BEGIN
-- Se existir, lançar um erro e impedir a exclusão
RAISERROR('Não é possível excluir o produto porque ele possui estoque
disponível.', 16, 1);
RETURN;
END
-- Se não houver estoque disponível, proceder com a exclusão
DELETE FROM dbo.Product
WHERE ProductId IN (SELECT ProductId FROM deleted);
END;
GO
```