

Faculdade Estácio - Polo Curitiba - Centro

Curso: Desenvolvimento Full Stack

Disciplina: Vamos manter as informações

Número da Turma: RPG0015

Semestre Letivo: 3

Integrante: Mariana Lucas Fernandes Onório

Repositório: <https://github.com/MariLFO/estacio-mundo3-missao-nivel-2>

Sumário:

Faculdade Estácio - Polo Curitiba - Centro	1
Sumário:	1
1. Título da Prática:	2
2. Objetivos da Prática:	2
3. Procedimento nº1 – Criando o Banco de Dados:	2
3.1. Códigos do roteiro: 1º Procedimento:	2
Arquivo: procedimento1_banco-de-dados.sql	2
3.2. Resultados da execução dos códigos:	3
3.3. Análise e Conclusão:	4
a) Como são implementadas as diferentes cardinalidades, basicamente 1X1, 1XN ou NxN, em um banco de dados relacional?	4
b) Que tipo de relacionamento deve ser utilizado para representar o uso de herança em bancos de dados relacionais?	4
c) Como o SQL Server Management Studio permite a melhoria da produtividade nas tarefas relacionadas ao gerenciamento do banco de dados?	4
4. Procedimento nº2 – Alimentando a Base:	5
4.1. Códigos do roteiro: 2º Procedimento:	5
Arquivo: procedimento2_alimentando-a-base.sql	5
4.2. Resultados da execução dos códigos:	8
4.3. Análise e Conclusão:	8
a) Quais as diferenças no uso de sequence e identity?	8
b) Qual a importância das chaves estrangeiras para a consistência do banco?	8
c) Quais operadores do SQL pertencem à álgebra relacional e quais são definidos no cálculo relacional?	8
d) Como é feito o agrupamento em consultas, e qual requisito é obrigatório?	8
5. Conclusão:	9

1. Título da Prática:

RPG0015 - Vamos manter as informações!

Modelagem e implementação de um banco de dados simples, utilizando como base o SQL Server.

2. Objetivos da Prática:

1. Identificar os requisitos de um sistema e transformá-los no modelo adequado.
2. Utilizar ferramentas de modelagem para bases de dados relacionais.
3. Explorar a sintaxe SQL na criação das estruturas do banco (DDL).
4. Explorar a sintaxe SQL na consulta e manipulação de dados (DML)
5. No final do exercício, o aluno terá vivenciado a experiência de modelar a base de dados para um sistema simples, além de implementá-la, através da sintaxe SQL, na plataforma do SQL Server.

3. Procedimento nº1 – Criando o Banco de Dados

3.1. Códigos do roteiro: 1º Procedimento

Arquivo: [procedimento1_banco-de-dados.sql](#)

```
USE [loja]
GO

CREATE SEQUENCE [dbo].[PessoaSequence]
AS INTEGER
START WITH 1
INCREMENT BY 1

CREATE TABLE Usuario(
    idUsuario INTEGER NOT NULL IDENTITY,
    login VARCHAR(255),
    senha VARCHAR(255),
    PRIMARY KEY(idUsuario));

CREATE TABLE Pessoa(
    idPessoa INTEGER NOT NULL PRIMARY KEY CLUSTERED DEFAULT (NEXT VALUE FOR dbo.PessoaSequence),
    nome VARCHAR(255) NOT NULL,
    logradouro VARCHAR(255),
    cidade VARCHAR(255),
    estado VARCHAR(2),
    telefone VARCHAR(11),
    email VARCHAR(255));
```

```
CREATE TABLE PessoaFisica(
    idPessoa INTEGER PRIMARY KEY CLUSTERED,
    CPF VARCHAR(11) NOT NULL,
    CONSTRAINT FK_PessoaFisica_Pessoa FOREIGN KEY (idPessoa) REFERENCES dbo.Pessoa (idPessoa));

CREATE TABLE PessoaJuridica(
    idPessoa INTEGER PRIMARY KEY CLUSTERED,
    CNPJ VARCHAR(14) NOT NULL,
    CONSTRAINT FK_PessoaJuridica_Pessoa FOREIGN KEY (idPessoa) REFERENCES dbo.Pessoa (idPessoa));

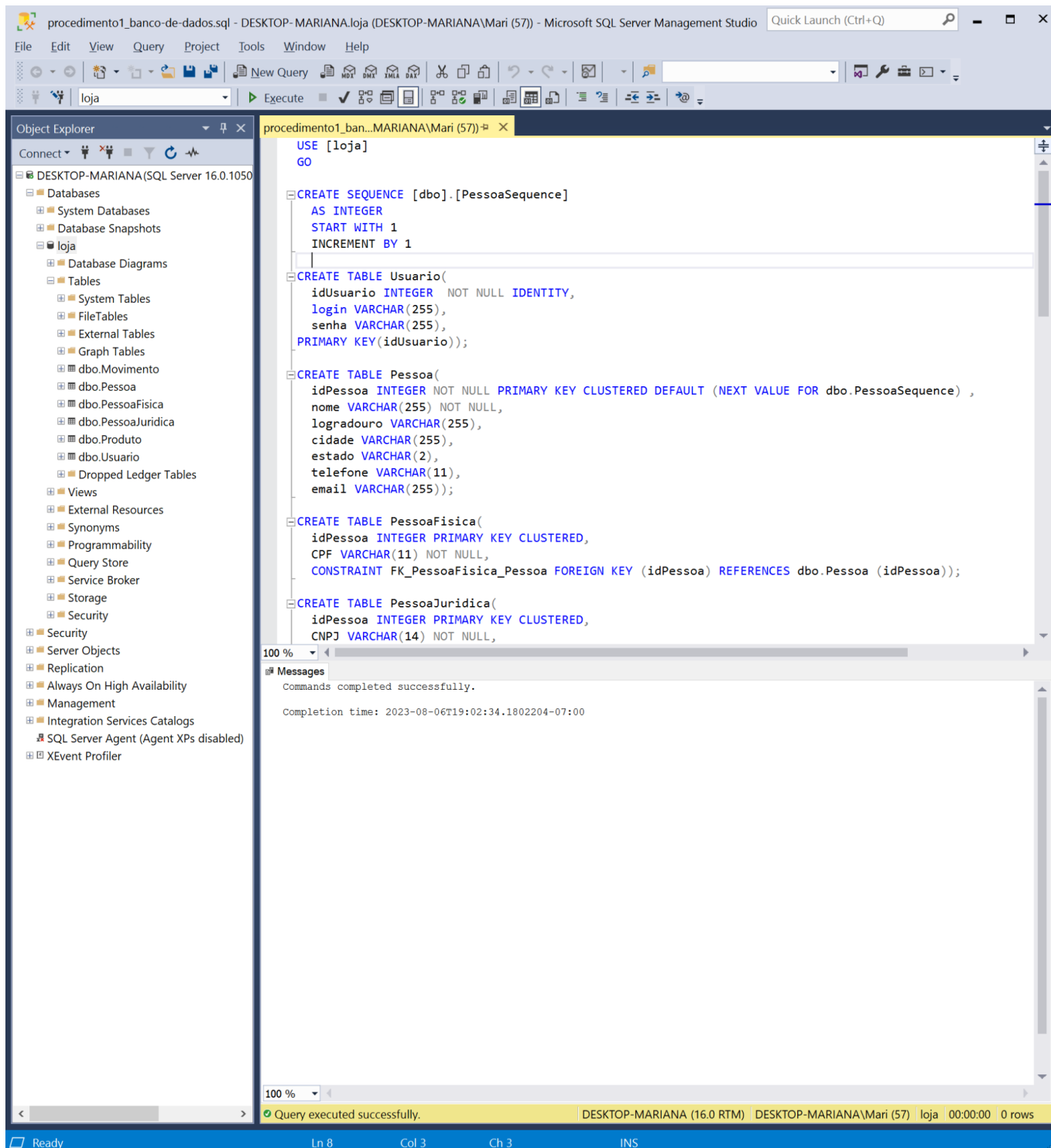
CREATE TABLE Produto(
    idProduto INTEGER NOT NULL IDENTITY,
    nome VARCHAR(255) NOT NULL,
    quantidade INTEGER NOT NULL,
    precoVenda NUMERIC NOT NULL,
    PRIMARY KEY(idProduto));

CREATE TABLE Movimento(
    idMovimento INTEGER NOT NULL IDENTITY,
    Usuario_idUsuario INTEGER NOT NULL,
    Pessoa_idPessoa INTEGER NOT NULL,
    Produto_idProduto INTEGER NOT NULL,
    quantidade INTEGER NOT NULL,
    tipo CHAR(1) NOT NULL,
    valorUnitario NUMERIC NOT NULL,
    PRIMARY KEY(idMovimento),
    FOREIGN KEY(Produto_idProduto)
        REFERENCES Produto(idProduto),
    FOREIGN KEY(Pessoa_idPessoa)
        REFERENCES Pessoa(idPessoa),
    FOREIGN KEY(Usuario_idUsuario)
        REFERENCES Usuario(idUsuario));

CREATE INDEX Movimento_FKIndex1 ON Movimento (Produto_idProduto);
CREATE INDEX Movimento_FKIndex2 ON Movimento (Pessoa_idPessoa);
CREATE INDEX Movimento_FKIndex3 ON Movimento (Usuario_idUsuario);

CREATE INDEX IFK_ItemMovimentado ON Movimento (Produto_idProduto);
CREATE INDEX IFK_Responsavel ON Movimento (Pessoa_idPessoa);
CREATE INDEX IFK_Operador ON Movimento (Usuario_idUsuario);
GO
```

3.2. Resultados da execução dos códigos:



The screenshot displays the Microsoft SQL Server Management Studio interface. The title bar indicates the file is 'procedimento1_banco-de-dados.sql' and the server is 'DESKTOP-MARIANA\loja (DESKTOP-MARIANA\Mari (57))'. The Object Explorer on the left shows the database structure, including tables like 'dbo.Usuario', 'dbo.Pessoa', 'dbo.PessoaFisica', and 'dbo.PessoaJuridica'. The main query editor contains the following SQL code:

```
USE [loja]
GO

CREATE SEQUENCE [dbo].[PessoaSequence]
AS INTEGER
START WITH 1
INCREMENT BY 1

CREATE TABLE Usuario(
    idUsuario INTEGER NOT NULL IDENTITY,
    login VARCHAR(255),
    senha VARCHAR(255),
    PRIMARY KEY(idUsuario));

CREATE TABLE Pessoa(
    idPessoa INTEGER NOT NULL PRIMARY KEY CLUSTERED DEFAULT (NEXT VALUE FOR dbo.PessoaSequence) ,
    nome VARCHAR(255) NOT NULL,
    logradouro VARCHAR(255),
    cidade VARCHAR(255),
    estado VARCHAR(2),
    telefone VARCHAR(11),
    email VARCHAR(255));

CREATE TABLE PessoaFisica(
    idPessoa INTEGER PRIMARY KEY CLUSTERED,
    CPF VARCHAR(11) NOT NULL,
    CONSTRAINT FK_PessoaFisica_Pessoa FOREIGN KEY (idPessoa) REFERENCES dbo.Pessoa (idPessoa));

CREATE TABLE PessoaJuridica(
    idPessoa INTEGER PRIMARY KEY CLUSTERED,
    CNPJ VARCHAR(14) NOT NULL,
```

The Messages pane at the bottom shows the execution results:

```
Messages
Commands completed successfully.
Completion time: 2023-08-06T19:02:34.1802204-07:00
```

The status bar at the bottom indicates 'Query executed successfully.' and shows the current position in the query: 'Ln 8 Col 3 Ch 3 INS'.

3.3. Análise e Conclusão:

- a) Como são implementadas as diferentes cardinalidades, basicamente 1X1, 1XN ou NxN, em um banco de dados relacional?

Resposta: A cardinalidade 1x1 (um-para-um) em um banco de dados relacional se apresenta quando uma entidade se relaciona com apenas uma entrada de outra entidade. Por exemplo, em um hospital, cada médico (1) tem um (e somente um) crachá (1), e cada crachá pertence a um (e somente um) médico.

A cardinalidade 1xN (um-para-muitos) significa que uma entrada se relaciona com muitas ocorrências de outra tabela, mas que a recíproca não é verdadeira, por exemplo, um funcionário (1) possui vários dependentes (N), mas cada dependente pertence a apenas um funcionário.

Para a cardinalidade NxN (muitos-para-muitos) entende-se que podem existir múltiplas entradas associadas a múltiplas ocorrências em outra tabela. Por exemplo, um aluno (N) pode cursar várias disciplinas (N), e uma disciplina pode ter vários alunos.

- b) Que tipo de relacionamento deve ser utilizado para representar o uso de herança em bancos de dados relacionais?

Resposta: Em bancos de dados relacionais, representamos o uso de herança através da especialização/generalização.

- c) Como o SQL Server Management Studio permite a melhoria da produtividade nas tarefas relacionadas ao gerenciamento do banco de dados?

Resposta: Ele permite rodar queries diretamente através de um console além de proporcionar funcionalidades como views, stored procedures, triggers, functions e índices.

4. Procedimento nº2 – Alimentando a Base

4.1. Códigos do roteiro: 2º Procedimento

Arquivo: [procedimento2_alimentando-a-base.sql](#)

```
-- Seleciona a base de dados:
USE [loja]
GO

-- Declara variável que será utilizada múltiplas vezes no código abaixo:
DECLARE @idPessoa INT;

-- Insere usuários:
INSERT INTO Usuario
([login],[senha])
VALUES ('op1', 'op1'), ('op2', 'op2')
GO

-- Insere produtos:
INSERT INTO Produto (nome, quantidade, precoVenda)
VALUES ('Banana', 100, 5.0),
       ('Laranja', 500, 2.0),
       ('Laranja', 800, 4.0);
GO

-- Insere pessoa física:
SET @idPessoa = NEXT VALUE FOR dbo.PessoaSequence;

INSERT INTO Pessoa (idPessoa, nome, logradouro, cidade, estado, telefone, email)
VALUES (@idPessoa, 'Joao', 'Rua 12, casa 3, Quitanda', 'Riacho do Sul', 'PA', '1111-1111',
'joao@riacho.com');

INSERT INTO PessoaFisica (idPessoa, CPF)
VALUES (@idPessoa, '11111111111');

-- Insere pessoas jurídica:
SET @idPessoa = NEXT VALUE FOR dbo.PessoaSequence;

INSERT INTO Pessoa (idPessoa, nome, logradouro, cidade, estado, telefone, email)
VALUES (@idPessoa, 'JJC', 'Rua 11, Centro', 'Riacho do Norte', 'PA', '1212-1212', 'jjc@riacho.com');

INSERT INTO PessoaJuridica (idPessoa, CNPJ)
VALUES (@idPessoa, '22222222222222');
```

```

-- Insere movimentações:
INSERT INTO Movimento (Usuario_idUsuario, Pessoa_idPessoa, Produto_idProduto, quantidade, tipo,
valorUnitario)
VALUES (1, 1, 1, 20, 'S', 4.00),
      (1, 1, 2, 15, 'S', 2.00),
      (2, 1, 2, 10, 'S', 3.00),
      (1, 2, 2, 15, 'E', 5.00),
      (1, 2, 3, 20, 'E', 4.00);

-- Lista usuários:
SELECT * FROM Usuario
GO

-- Lista dados completos de pessoas físicas:
SELECT * FROM Pessoa p JOIN PessoaFisica pf ON p.idPessoa = pf.idPessoa;
GO

-- Lista dados completos de pessoas jurídicas:
SELECT * FROM Pessoa p JOIN PessoaJuridica pj ON p.idPessoa = pj.idPessoa;
GO

-- Lista movimentações de entrada, com produto, fornecedor, quantidade, preço unitário e valor total:
SELECT Produto.nome AS produto,
      Pessoa.nome AS fornecedor,
      Movimento.quantidade,
      Movimento.valorUnitario,
      Movimento.quantidade * Movimento.valorUnitario AS "valor total"
FROM Movimento
      JOIN Produto ON Movimento.Produto_idProduto = Produto.idProduto
      JOIN Pessoa ON Movimento.Pessoa_idPessoa = Pessoa.idPessoa
WHERE tipo = 'E';
GO

-- Lista movimentações de saída, com produto, comprador, quantidade, preço unitário e valor total:
SELECT Produto.nome AS produto,
      Pessoa.nome AS comprador,
      Movimento.quantidade,
      Movimento.valorUnitario,
      Movimento.quantidade * Movimento.valorUnitario AS "valor total"
FROM Movimento
      JOIN Produto ON Movimento.Produto_idProduto = Produto.idProduto
      JOIN Pessoa ON Movimento.Pessoa_idPessoa = Pessoa.idPessoa
WHERE tipo = 'S';
GO

```

```

-- Lista valor total das entradas agrupadas por produto:
SELECT Produto.nome AS produto,
       SUM(Movimento.quantidade * Movimento.valorUnitario) AS "valor total entradas"
FROM Movimento
     JOIN Produto ON Movimento.Produto_idProduto = Produto.idProduto
WHERE tipo = 'E'
GROUP BY Produto.nome;
GO

-- Lista valor total das saídas agrupadas por produto:
SELECT Produto.nome AS produto,
       SUM(Movimento.quantidade * Movimento.valorUnitario) AS "valor total saidas"
FROM Movimento
     JOIN Produto ON Movimento.Produto_idProduto = Produto.idProduto
WHERE tipo = 'S'
GROUP BY Produto.nome;
GO

-- Lista operadores que não efetuaram movimentações de entrada (compra):
SELECT Usuario.login AS "operador sem compra"
FROM Usuario
     LEFT JOIN (SELECT DISTINCT Usuario_idUsuario FROM Movimento WHERE tipo = 'E') m ON Usuario.idUsuario =
m.Usuario_idUsuario
WHERE m.Usuario_idUsuario IS NULL;

-- Lista valor total de entrada, agrupado por operador:
SELECT Usuario.login AS operador,
       SUM(Movimento.quantidade * Movimento.valorUnitario) AS "valor total entradas"
FROM Movimento
     JOIN Usuario ON Movimento.Usuario_idUsuario = Usuario.idUsuario
WHERE tipo = 'E'
GROUP BY Usuario.login;

-- Lista valor total de saída, agrupado por operador:
SELECT Usuario.login AS operador,
       SUM(Movimento.quantidade * Movimento.valorUnitario) AS "valor total saidas"
FROM Movimento
     JOIN Usuario ON Movimento.Usuario_idUsuario = Usuario.idUsuario
WHERE tipo = 'S'
GROUP BY Usuario.login;
GO

-- Lista valor médio de venda por produto, utilizando média ponderada:
WITH VendasPorProduto AS (
    SELECT Produto_idProduto,
           SUM(quantidade * valorUnitario) / SUM(quantidade) AS valorMedioVendaPorProduto
    FROM Movimento
    WHERE tipo = 'S'
    GROUP BY Produto_idProduto)
SELECT Produto.nome AS produto,
       VendasPorProduto.valorMedioVendaPorProduto AS "valor medio de venda por produto (media
ponderada)"
FROM VendasPorProduto
     JOIN Produto ON VendasPorProduto.Produto_idProduto = Produto.idProduto;
GO

```


4.2. Resultados da execução dos códigos:

The screenshot displays the Microsoft SQL Server Enterprise Manager interface. The left pane shows the 'Object Explorer' with the 'loja' database selected. The right pane shows a SQL script titled 'procedimento2_alimentando-a-base.sql' with the following content:

```
-- Seleciona a base de dados:
USE [loja]
GO

-- Declara variável que será utilizada múltiplas vezes no código abaixo:
DECLARE @idPessoa INT;

-- Inserir usuários:
INSERT INTO Usuario
([login],[senha])
VALUES ('op1', 'op1'), ('op2', 'op2')
GO

-- Inserir produtos:
INSERT INTO Produto (nome, quantidade, precoVenda)
VALUES ('Banana', 100, 5.0),
('Laranja', 500, 2.0),
('Laranja', 800, 4.0);
GO

-- Inserir pessoa física:
DECLARE @idPessoa INT;
```

Below the script, the 'Results' pane shows the output of the execution. The status bar at the bottom indicates 'Query executed successfully.' and '18 rows'.

idUsuario	login	senha
1	op1	op1
2	op2	op2

idPessoa	nome	logradouro	cidade	estado	telefone	email	idPessoa	CPF
1	Joao	Rua 12, casa 3, Quitanda	Riacho do Sul	PA	1111-1111	joao@riacho.com	1	11111111111
2	JJC	Rua 11, Centro	Riacho do Norte	PA	1212-1212	jjc@riacho.com	2	22222222222

produto	fornecedor	quantidade	valorUnitario	valor total
Laranja	JJC	15	5	75
Laranja	JJC	20	4	80

produto	comprador	quantidade	valorUnitario	valor total
Banana	Joao	20	4	80
Laranja	Joao	15	2	30
Laranja	Joao	10	3	30

produto	valor total	entradas
Laranja	155	

produto	valor total	saidas
Banana	80	
Laranja	60	

operador	sem compra
op2	

operador	valor total	entradas
op1	155	

operador	valor total	saidas
op1	110	
op2	30	

produto	valor medio de venda por produto (media ponderada)
Banana	4.000000
Laranja	2.400000

4.3. Análise e Conclusão:

a) Quais as diferenças no uso de sequence e identity?

Resposta: A Identity é uma propriedade de uma coluna de uma tabela, portanto ela só pode ser usada na tabela em que foi definida, não podendo ser compartilhada por outras tabelas. A Sequence, por outro lado, é um objeto definido pelo usuário e não fica atrelado a nenhuma tabela específica, desta forma, podendo ser compartilhada por várias tabelas através do comando NEXT VALUE FOR.

b) Qual a importância das chaves estrangeiras para a consistência do banco?

Resposta: As chaves estrangeiras são identificadores únicos que conectam duas ou mais tabelas, fazendo referência às chaves primárias ou campos únicos de outras tabelas, garantindo consistência dos dados relacionados.

c) Quais operadores do SQL pertencem à álgebra relacional e quais são definidos no cálculo relacional?

Resposta: Alguns dos operadores do SQL que pertencem à álgebra relacional são: seleção, projeção, união, interseção, diferença, através de SELECT, WHERE, UNION, JOIN, =, <, >, etc.

d) Como é feito o agrupamento em consultas, e qual requisito é obrigatório?

Resposta: Os agrupamentos no SQL são feitos através do comando GROUP BY. É obrigatório uma função de agregação em uma coluna para produzir um resultado agrupado, além disso é necessário dizer a coluna responsável pela estratificação do resultado da função.

5. Conclusão:

Nessa Missão Prática, foi possível vivenciar a experiência de modelar e implementar uma base de dados relacional para um sistema simples, além de explorar a sintaxe SQL na criação e manipulação das estruturas do banco de dados. Além disso, foi possível praticar a utilização das ferramentas DBDesigner e SQL Server Management Studio, que facilitam o processo de desenvolvimento e gerenciamento das bases de dados.

Utilizar o DBDesigner permite uma fácil visualização de como os dados e tabelas se relacionam entre si através dos diagramas de entidade-relacionamento (DER), além disso ele também gera automaticamente os comandos SQL para a criação das tabelas, índices, chaves e relacionamentos a partir do DER, exportando para vários tipos de banco de dados diferentes. Apesar dessa facilidade, no entanto, é importante adquirir o conhecimento necessário do SQL e revisar o código gerado, pois o programa pode apresentar erros ou ineficiências, principalmente quando há relacionamentos mais complexos.

O SQL Server Management Studio permitiu a conexão e administração dos dados de forma centralizada, o que facilita executar os comandos SQL, visualizar e editar dados, criar e modificar tabelas, índices entre outras funcionalidades. Ele oferece várias ferramentas de testes, otimização, backup, restauração, importação e exportação, o que facilita o trabalho de um administrador de banco de dados. No entanto, a ferramenta pode ser complexa e confusa para iniciantes ou usuários menos experientes, pois possui muitas opções, menus, janelas e ferramentas, então requer prática e treinamento apropriado para extrair o máximo de suas funcionalidades.

No geral, houveram alguns desafios, como a necessidade de instalar e configurar as ferramentas, revisar os comandos SQL gerados e verificar a coerência dos modelos e diagramas, mas foi muito gratificante ver o resultado final e o aprendizado para utilização desses conceitos e ferramentas de uma forma mais prática.