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

Michael Rodrigues Dos Santos

Polo Manilha-Itaboraí

RPG0015-vamos Manter as informações – **Número da Turma**
9001 – 3° Semestre

Objetivo da Prática

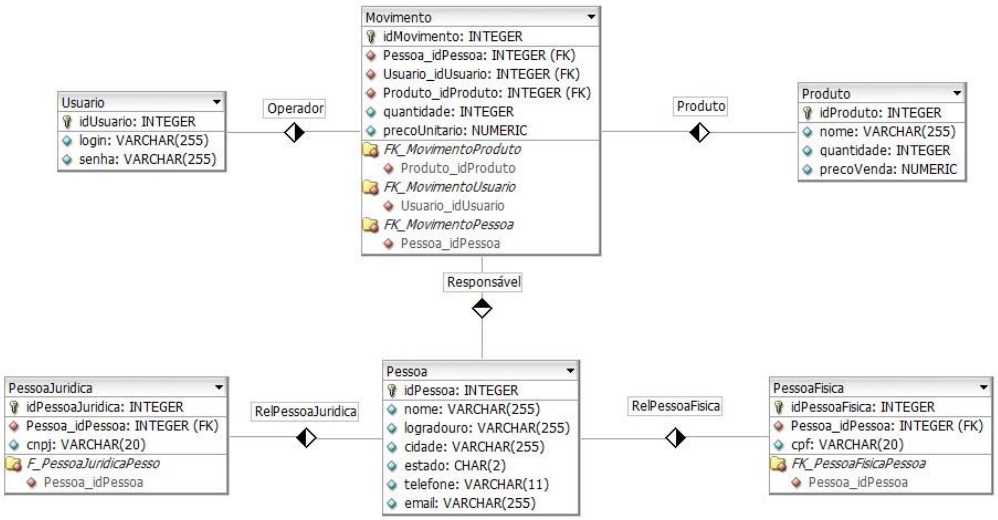
Reconhecer as exigências de um sistema e convertê-las no formato apropriado.

Empregar instrumentos de modelagem para bases de dados relacionais.

Investigar a sintaxe SQL durante a elaboração das estruturas do banco (DDL).

Explorar a sintaxe SQL na pesquisa e manipulação de dados (DML).

1º Procedimento | Criacão da modelagem do banco de dados usando a ferramenta DB Designer



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

a. Que tipo de relacionamento deve ser utilizado para representar o uso de herança em bancos de dados relacionais?

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

2º Criando as Tabelas

```
USE [master]
GO CREATE DATABASE; GO USE ; GO

CREATE TABLE [Produto] (
  idProduto INTEGER IDENTITY(1,1) NOT NULL,
nome VARCHAR(255) NOT NULL,   quantidade
INTEGER NOT NULL,   precoVenda NUMERIC(10,
2) NOT NULL,
  PRIMARY KEY(idProduto)
);
GO

CREATE TABLE [Pessoa] (
  idPessoa INTEGER IDENTITY(1,1) NOT NULL,
nome VARCHAR(255) NOT NULL,   logradouro
VARCHAR(255) NOT NULL,   cidade
VARCHAR(255) NOT NULL,



  estado CHAR(2) NOT NULL,
telefone VARCHAR(11),   email
VARCHAR(255),   PRIMARY
KEY(idPessoa)
);
GO

CREATE TABLE [Usuario] (   idUsuario INTEGER IDENTITY(1,1) NOT NULL,
login VARCHAR(255) NOT NULL UNIQUE,   senha VARCHAR(255) NOT NULL,
PRIMARY KEY(idUsuario)
```

```
);
GO

CREATE TABLE [PessoaJuridica] (
  idPessoaJuridica INTEGER IDENTITY(1,1) NOT NULL,
  Pessoa_idPessoa INTEGER NOT NULL,
cnpj VARCHAR(20) NOT NULL,   PRIMARY
KEY(idPessoaJuridica),
  FOREIGN KEY(Pessoa_idPessoa) REFERENCES Pessoa(idPessoa)
);
GO

CREATE INDEX F_PessoaJuridicaPesso ON PessoaJuridica (Pessoa_idPessoa);
GO

CREATE TABLE [PessoaFisica] (
  idPessoaFisica INTEGER IDENTITY(1,1) NOT NULL,
  Pessoa_idPessoa INTEGER NOT NULL,
cpf VARCHAR(20) NOT NULL,   PRIMARY
KEY(idPessoaFisica),
  FOREIGN KEY(Pessoa_idPessoa) REFERENCES Pessoa(idPessoa)
);
GO

CREATE INDEX FK_PessoaFisicaPessoa ON PessoaFisica (Pessoa_idPessoa);
GO

CREATE TABLE [Movimento] (
  idMovimento INTEGER IDENTITY(1,1) NOT NULL,
  Pessoa_idPessoa INTEGER NOT NULL,
  Usuario_idUsuario INTEGER NOT NULL,
Produto_idProduto INTEGER NOT NULL,
quantidade INTEGER NOT NULL,   precoUnitario
NUMERIC(10, 2) NOT NULL,
  PRIMARY KEY(idMovimento),
  FOREIGN KEY(Produto_idProduto) REFERENCES Produto(idProduto),
  FOREIGN KEY(Usuario_idUsuario) REFERENCES Usuario(idUsuario),
  FOREIGN KEY(Pessoa_idPessoa) REFERENCES Pessoa(idPessoa)
);
GO

CREATE INDEX FK_MovimentoProduto ON Movimento (Produto_idProduto);
GO

CREATE INDEX FK_MovimentoUsuario ON Movimento (Usuario_idUsuario);
GO

CREATE INDEX FK_MovimentoPessoa ON Movimento (Pessoa_idPessoa); GO
```

2º Procedimento | Alimentando a Base

Cadastrando Usuários:

```
INSERT INTO [Usuario] (login ,senha) VALUES ('op1','op1') GO
```

```
INSERT INTO [Usuario] (login ,senha) VALUES ('op2','op2') GO INSERT INTO [Usuario]
(login ,senha) VALUES ('op3','op3') GO
```

Consultando Usuários:

Resultados | Mensagens

	idUsuario	login	senha
1	1	op1	op1
2	3	op2	op2
3	5	op3	op3

```
SELECT * FROM [Usuario]
GO
```

Adicionando Produtos:

```
INSERT INTO [Produto] ([nome],[quantidade], [precoVenda])
VALUES
('Banana','100','5'),
('Laranja','500','2'),
('Manga','800','4'),
('Maracuja','200','3.30');
GO
```

Selecionando Produtos:

Resultados | Mensagens

	idProduto	nome	quantidade	precoVenda
1	1	Banana	100	5.00
2	2	Laranja	500	2.00
3	3	Manga	800	4.00
4	4	Maracuja	200	3.30

```
SELECT * FROM [Produto]
GO
```

Selecionando todas as Pessoas:

100 % | Resultados | Mensagens

	idPessoa	nome	logradouro	cidade	estado	telefone	email
1	1	João Silva	Rua Exemplo 1	Cidade Exemplo	RJ	12345678901	joao.silva@email.com
2	2	Maria Souza	Avenida Exemplo 2	Cidade Exemplo	SC	10987654321	maria.souza@email.com
3	3	Marcio Silva	Rua Exemplo 3	Cidade Exemplo	RS	12345678901	marciosilva@example.com
4	4	Carlos Souza	Travessa Exemplo 4	Cidade Exemplo	SP	09876543210	carlossouza@example.com

```
Listar todas as pessoas
SELECT * FROM [Pessoa] GO
```

Selecionando Pessoas Físicas:

	idPessoa	nome	logradouro	cidade	estado	telefone	email	cpf
1	1	João Silva	Rua Exemplo 1	Cidade Exemplo	RJ	12345678901	joao.silva@email.com	22222221121
2	4	Carlos Souza	Travessa Exemplo 4	Cidade Exemplo	SP	09876543210	carlossouza@example.com	33334412231

```
Consulta de Pessoas Físicas
SELECT Pessoa.idPessoa, Pessoa.nome, Pessoa.logradouro, Pessoa.cidade,
Pessoa.estado, Pessoa.telefone, Pessoa.email, PF.cpf
FROM Pessoa AS Pessoa JOIN PessoaFisica AS PF ON Pessoa.idPessoa =
PF.Pessoa_idPessoa; GO
```

Selecionando Pessoas Juridicas:

	idPessoa	nome	logradouro	cidade	estado	telefone	email	cnpj
1	2	Maria Souza	Avenida Exemplo 2	Cidade Exemplo	SC	10987654321	maria.souza@email.com	888888123123
2	3	Marcio Silva	Rua Exemplo 3	Cidade Exemplo	RS	12345678901	marciosilva@example.com	999999123121

```
Consulta de Pessoas Jurídicas
SELECT Pessoa.idPessoa, Pessoa.nome, Pessoa.logradouro, Pessoa.cidade,
Pessoa.estado, Pessoa.telefone, Pessoa.email, PJ.cnpj
FROM Pessoa AS Pessoa JOIN PessoaJuridica AS PJ ON Pessoa.idPessoa =
PJ.Pessoa_idPessoa;

GO
```

Inserindo Movimentos de Entrada / Saida:

```
INSERT INTO [Movimento] (Usuario_idUsuario, Pessoa_idPessoa , Produto_idProduto,
quantidade, tipo, precoUnitario)
VALUES
(1, 2, 2, 8, 'S', 2),
(2, 3, 1, 2, 'E', 5),
(1, 4, 1, 12, 'E', 5),
(1, 4, 2, 12, 'S', 2);

GO
```

Inserindo Movimentos de Entrada / Saida:

100 %

Resultados | Mensagens

	idMovimento	Pessoa_idPessoa	Usuario_idUsuario	Produto_idProduto	tipo	quantidade	precoUnitario
1	1	2	1	2	S	8	2.00
2	2	3	2	1	E	2	5.00
3	3	4	1	1	E	12	5.00

```
SELECT * FROM Movimento;
GO
```

Consultas sobre os dados inseridos:

a) Dados completos de pessoas físicas.

```
SELECT Pessoa.idPessoa, Pessoa.nome, Pessoa.logradouro, Pessoa.cidade,
Pessoa.estado, Pessoa.telefone, Pessoa.email, PF.cpf
FROM Pessoa AS Pessoa JOIN PessoaFisica AS PF ON Pessoa.idPessoa =
PF.Pessoa_idPessoa;
GO
```

b) Dados completos de pessoas jurídicas.

```
SELECT Pessoa.idPessoa, Pessoa.nome, Pessoa.logradouro, Pessoa.cidade,
Pessoa.estado, Pessoa.telefone, Pessoa.email, PJ.cnpj
FROM Pessoa AS Pessoa JOIN PessoaJuridica AS PJ ON Pessoa.idPessoa =
PJ.Pessoa_idPessoa;
GO
```

e) Movimentações de entrada, com produto, fornecedor, quantidade, preço unitário e valor total.

```
SELECT
    pessoa.nome AS NomePessoa,
    produto.nome AS NomeProduto,
mov.quantidade,
    mov.precoUnitario AS ValorUnitario,
    mov.quantidade * mov.precoUnitario AS valorTotal
FROM Movimento mov
JOIN Produto produto ON mov.Produto_idProduto = produto.idProduto
JOIN Pessoa pessoa ON mov.Pessoa_idPessoa = pessoa.idPessoa
WHERE mov.tipo = 'E';
GO
```

f) Movimentações de saída, com produto, comprador, quantidade, preço unitário e valor total.

```
    SELECT
        pe.nome AS NomePessoa,
    produto.nome AS NomeProduto,
    mov.quantidade,
        mov.precoUnitario AS ValorUnitario,
        mov.quantidade * mov.precoUnitario AS valorTotal
    FROM Movimento mov
    JOIN Produto produto ON mov.Produto_idProduto = produto.idProduto
    JOIN Pessoa pe ON mov.Pessoa_idPessoa = pe.idPessoa
    WHERE mov.tipo = 'S'; GO
```

g) Valor total das entradas agrupadas por produto.

```
Total de Entradas agrupadas por produtos
SELECT Produto_idProduto, SUM(precoUnitario) AS TotalPrecoUnitario FROM
Movimento
WHERE tipo = 'E'
GROUP BY Produto_idProduto;
GO
```

h) Valor total das saídas agrupadas por produto.

```
SELECT
produto.nome AS NomeProduto, mov.Produto_idProduto,
SUM(mov.quantidade * mov.precoUnitario) AS TotalValor
FROM Movimento mov
JOIN Produto produto ON mov.Produto_idProduto = produto.idProduto
WHERE mov.tipo = 'S'
GROUP BY mov.Produto_idProduto, produto.nome;
GO
```

i) Operadores que não efetuaram movimentações de entrada (compra).

```
SELECT DISTINCT usuarios.*
FROM Usuario usuarios
LEFT JOIN Movimento mov ON usuarios.idUsuario = mov.Usuario_idUsuario AND
mov.tipo = 'S'
WHERE mov.Usuario_idUsuario IS NULL;
GO
```

j) Valor total de entrada, agrupado por operador.

```
SELECT
    Usuario_idUsuario,
    SUM(quantidade * precoUnitario) AS TotalValor
FROM Movimento
WHERE tipo = 'E'
GROUP BY Usuario_idUsuario;
GO
```

k) Valor total de saída, agrupado por operador.

```
SELECT
    Usuario_idUsuario,
    SUM(quantidade * precoUnitario) AS TotalValor
FROM Movimento
WHERE tipo = 'S'
```

```
GROUP BY Usuario_idUsuario;
GO
```

l) Valor médio de venda por produto, utilizando média ponderada.

```
SELECT
    produto.nome AS NomeProduto,
mov.Produto_idProduto,
    AVG(mov.precoUnitario) AS MediaPrecoUnitario
FROM Movimento mov
JOIN Produto produto ON mov.Produto_idProduto = produto.idProduto
GROUP BY mov.Produto_idProduto, produto.nome;
GO
```

Quais as diferenças no uso de sequence e identity?

`SEQUENCE` e `IDENTITY` são duas abordagens diferentes para gerar valores únicos em bancos de dados relacionais:

- **SEQUENCE:**

- É um objeto do banco de dados que gera uma sequência de números.

- Pode ser compartilhado por várias tabelas.

- Oferece maior flexibilidade no controle e manipulação dos valores gerados.

- **IDENTITY:**

- É uma propriedade aplicada a uma coluna específica de uma tabela.

- Gera valores exclusivos automaticamente para essa coluna.

- Vinculado diretamente à coluna e é específico para a tabela.

a. Qual a importância das chaves estrangerias para a consistência do banco?

O uso de chaves estrangeiras assegura a integridade referencial, promove relacionamentos claros entre tabelas, evita dados órfãos, mantém a consistência durante alterações e facilita operações de atualização e exclusão.

b. Quais operadores do SQL pertencem à álgebra relacional e quais são definidos no cálculo relacional?

- Ambas as álgebra e cálculo relacional utilizam os operadores de projeção e seleção.A álgebra relacional possui operadores adicionais, como união, interseção, diferença e produto cartesiano, que não têm equivalentes diretos no cálculo relacional.

c. Como é feito o agrupamento em consultas, e qual requisito é obrigatório?

- **coluna1** e **coluna2** são as colunas pelas quais os resultados serão agrupados.
- **COUNT(*)** é uma função de agregação que conta o número de linhas em cada grupo.

Requisitos obrigatórios:

- Todas as colunas listadas após **GROUP BY** devem fazer parte da lista de seleção ou devem ser usadas com funções de agregação, como COUNT, SUM, AVG, etc.
- Isso garante que o SQL saiba como agregar os dados para cada grupo.

Conclusão

A Missão abordaram diversos aspectos ao bancos de dados e SQL. Foi destacada a importância das chaves estrangeiras para manter a consistência dos dados, garantindo a integridade referencial e facilitando relacionamentos entre tabelas. As diferenças entre os operadores da álgebra relacional e do cálculo relacional no SQL foram esclarecidas, fornecendo insights sobre as operações de manipulação de dados.

Além disso, foram abordados conceitos como o uso de `SEQUENCE` e `IDENTITY` para geração de valores únicos, e diferentes abordagens para modelar herança em bancos de

dados relacionais. A análise crítica destaca a importância de escolher a abordagem certa com base nos requisitos específicos do sistema.

A discussão sobre agrupamento com `GROUP BY` enfatizou a necessidade de incluir todas as colunas de agrupamento na lista de seleção ou usá-las com funções de agregação. Esse aspecto é crucial para garantir que o SQL possa agregar dados de maneira consistente.

Em resumo, as discussões proporcionaram uma compreensão abrangente de conceitos fundamentais de bancos de dados e SQL, destacando a importância da modelagem correta, integridade dos dados e escolhas adequadas para atender aos requisitos específicos de cada aplicação.