POLO CHAPADA - MANAUS - AM/UNIVERSIDADE ESTÁCIO DE SÁ

Missão Prática | Nível 2 | Mundo 3

Curso: Desenvolvimento Full Stack
Disciplina Nível 2: RPG0015 – Vamos manter as informações!
Número da Turma: 2024.2
Semestre Letivo: Mundo-3
Aluno: Gilvan Júnior Nascimento Gonçalves
Matrícula: 202304560188
URL GIT:

1º Título da Prática: Criando o Banco de Dados

2º Objetivo da Prática:

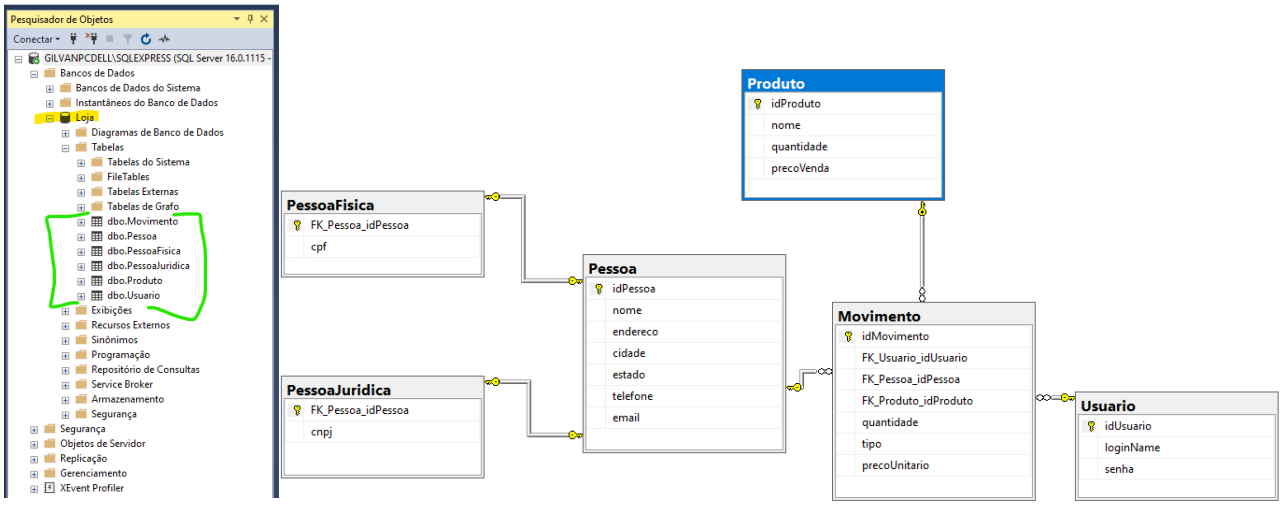
Identificar os requisitos de um sistema e transformá-los no modelo adequado.
Utilizar ferramentas de modelagem para bases de dados relacionais.
Explorar a sintaxe SQL na criação das estruturas do banco (DDL).
Explorar a sintaxe SQL na consulta e manipulação de dados (DML) No final do exercício, o aluno 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.

Tabelas a serem Criadas:

1. Usuário
2. Pessoa
3. Pessoa física;
4. Pessoa jurídica;
5. Produto;
6. Movimentação.

1º Procedimento | Criando o Banco de Dados.

Imagem 1: Diagrama de Classe - Entidade-Relacionamento (DER)



Definir o modelo de dados para um sistema com as características apresentadas nos tópicos seguintes:

a. Deve haver um cadastro de **usuários** para acesso ao sistema, os quais irão atuar como operadores para a compra e venda de produtos.

```
CREATE TABLE Usuario(
idUsuario INTEGER NOT NULL IDENTITY,
loginName VARCHAR(20) NOT NULL,
senha VARCHAR(20) NOT NULL,
CONSTRAINT CPK_Usuario PRIMARY KEY CLUSTERED(idUsuario ASC)
);
GO
```

b. Deve haver um cadastro de pessoas físicas e pessoas jurídicas, com os dados básicos de identificação, localização e contato, diferenciando-se apenas pelo uso de CPF ou CNPJ.

```
CREATE TABLE PessoaFisica(
FK_Pessoa_idPessoa INTEGER NOT NULL,
cpf VARCHAR(11) NOT NULL,
CONSTRAINT CPK_PessoaFisica PRIMARY KEY CLUSTERED(FK_Pessoa_idPessoa ASC),
CONSTRAINT CFK_Pessoa_PessoaFisica FOREIGN KEY(FK_Pessoa_idPessoa) REFERENCES
Pessoa(idPessoa)
ON UPDATE CASCADE
ON DELETE CASCADE
);
GO

CREATE TABLE PessoaJuridica(
FK_Pessoa_idPessoa INTEGER NOT NULL,
cnpj VARCHAR(14) NOT NULL,
CONSTRAINT CPK_PessoaJuridica PRIMARY KEY CLUSTERED(FK_Pessoa_idPessoa ASC),
CONSTRAINT CFK_Pessoa_PessoaJuridica FOREIGN KEY(FK_Pessoa_idPessoa) REFERENCES
Pessoa(idPessoa)
ON UPDATE CASCADE
ON DELETE CASCADE
);
GO
```

c. Deve haver um cadastro de produtos, contendo identificador, nome, quantidade e preço de venda.

```
CREATE TABLE Produto(
idProduto INTEGER NOT NULL IDENTITY,
nome VARCHAR(255) NOT NULL,
quantidade INTEGER,
precoVenda decimal(18,2),
CONSTRAINT CPK_Produto PRIMARY KEY CLUSTERED(idProduto ASC)
);
GO
```

d. Definir uma sequence para geração dos identificadores de pessoa, dado o relacionamento 1x1 com pessoa física ou jurídica.

```
USE Loja;
GO

CREATE SEQUENCE orderPessoa
AS INT
START WITH 1
INCREMENT BY 1;
```

a. Os operadores (usuários) poderão efetuar **movimentos** de compra para um determinado produto, sempre de uma pessoa jurídica, indicando a quantidade de produtos e preço unitário.

```
CREATE TABLE Movimento (
idMovimento INTEGER NOT NULL IDENTITY,
FK_Usuario_idUsuario INTEGER NOT NULL,
FK_Pessoa_idPessoa INTEGER NOT NULL,
FK_Produto_idProduto INTEGER NOT NULL,
quantidade INTEGER,
tipo CHAR(1),
precoUnitario NUMERIC,
CONSTRAINT CPK_Movimento PRIMARY KEY CLUSTERED(idMovimento ASC),
CONSTRAINT CFK_Usuario_Movimento FOREIGN KEY(FK_Usuario_idUsuario) REFERENCES
Usuario(idUsuario)
ON UPDATE CASCADE
ON DELETE CASCADE,
CONSTRAINT CFK_Pessoa_Movimento FOREIGN KEY(FK_Pessoa_idPessoa) REFERENCES Pessoa(idPessoa)
ON UPDATE CASCADE
ON DELETE CASCADE,
CONSTRAINT CFK_Produto_Movimento FOREIGN KEY(FK_Produto_idProduto) REFERENCES
Produto(idProduto)
ON UPDATE CASCADE
ON DELETE CASCADE
);
GO

CREATE TABLE Pessoa(
idPessoa INTEGER NOT NULL,
nome VARCHAR(255),
endereco VARCHAR(255),
cidade VARCHAR(255),
estado CHAR(2),
telefone VARCHAR(15),
email VARCHAR(255),
CONSTRAINT CPK_Pessoa PRIMARY KEY CLUSTERED(idPessoa ASC)
);
GO
```

Análise e Conclusão do 1º Procedimento:

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

No mundo dos bancos de dados relacionais, as cardinalidades 1:1, 1:N e N:N definem as relações entre as entidades, ditando quantas instâncias de uma entidade podem se associar a quantas instâncias de outra. Cada tipo de cardinalidade possui mecanismos específicos para sua implementação:

1:1 (Um para Um):

- **Chave Primária e Chave Estrangeira:** Nessa relação, cada entidade possui sua própria chave primária. Uma das entidades referencia a chave primária da outra através de uma chave estrangeira. Essa chave estrangeira torna-se obrigatória na tabela referenciada, garantindo que cada registro na tabela referenciada esteja associado a um único registro na tabela principal.
- **Exemplo:** Tabela Cliente (ID_Cliente, Nome, CPF) e Tabela Endereço (ID_Endereço, Rua, Cidade, ID_Cliente). Aqui, cada cliente possui apenas um endereço e cada endereço pertence a um único cliente. A tabela Endereço possui a chave estrangeira ID_Cliente que referencia a chave primária da tabela Cliente.

1:N (Um para Muitos):

- **Chave Primária e Chave Estrangeira:** Semelhante ao 1:1, a entidade principal possui sua chave primária, enquanto a entidade dependente possui chave primária própria e uma chave estrangeira que referencia a chave primária da principal. A chave estrangeira na tabela dependente também é obrigatória.
- **Exemplo:** Tabela Professor (ID_Professor, Nome, Departamento) e Tabela Turma (ID_Turma, Disciplina, ID_Professor). Um professor pode lecionar várias turmas, mas cada turma tem apenas um professor. A tabela Turma possui a chave estrangeira ID_Professor que referencia a chave primária da tabela Professor.

N:N (Muitos para Muitos):

- **Tabela de Junção:** Diferentemente das cardinalidades 1:1 e 1:N, aqui não há chave estrangeira em nenhuma das tabelas principais. Uma nova tabela é criada, chamada de tabela de junção, contendo chaves primárias de ambas as entidades e colunas adicionais se necessário.
- **Exemplo:** Tabela Aluno (ID_Aluno, Nome, Curso) e Tabela Disciplina (ID_Disciplina, Nome, Professor). Um aluno pode cursar várias disciplinas e uma disciplina pode ter vários alunos. A tabela de junção Aluno_Disciplina (ID_Aluno, ID_Disciplina) seria criada para relacionar os alunos com as disciplinas cursadas.

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

- Existem duas abordagens principais para implementar herança em bancos de dados relacionais:

1. **Tabela por classe** (ou tabela única): Nesta abordagem, todas as classes (ou tipos) na hierarquia de herança são representadas em uma única tabela. Esta tabela contém todos os atributos da hierarquia, sendo que para uma instância de uma classe específica, os atributos correspondentes às classes mais gerais são preenchidos com valores nulos.

2. **Tabela por subclasse** (ou tabela separada): Aqui, cada classe (ou tipo) na hierarquia de herança é representada por uma tabela separada. A tabela correspondente à classe mais geral contém os atributos comuns a todas as classes na hierarquia. Cada tabela subsequente na hierarquia contém seus próprios atributos exclusivos além dos atributos herdados.

   O relacionamento de generalização/especialização permite modelar essas hierarquias de forma que consultas e operações possam ser feitas eficientemente, levando em consideração a estrutura hierárquica e os relacionamentos entre as entidades (ou tabelas).

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

O SQL Server Management Studio (SSMS) é uma ferramenta robusta da Microsoft projetada para facilitar o gerenciamento de bancos de dados SQL Server. Ela melhora a produtividade em diversas tarefas relacionadas ao gerenciamento de bancos de dados de várias maneiras conforme alguns tópicos abaixo:

1. **Interface Gráfica Amigável:** O SSMS fornece uma interface gráfica intuitiva que permite aos administradores de banco de dados e desenvolvedores realizar tarefas complexas de forma mais eficiente do que usando apenas comandos SQL.

2. **Editor SQL Avançado**: Inclui um editor SQL poderoso com recursos como realce de sintaxe, sugestões automáticas, formatação de código e verificação de erros, facilitando a escrita e edição de consultas SQL complexas.

3. Gerenciamento de Servidores: Permite conectar e gerenciar facilmente vários servidores SQL Server a partir de uma única interface, possibilitando o monitoramento do status do servidor, configuração de propriedades e gerenciamento de segurança.

4. **Ferramentas de Desenvolvimento**: Oferece ferramentas integradas para desenvolvimento de bancos de dados, como o Designer de Tabelas, o Designer de

Consultas e o Designer de Procedimentos Armazenados, que simplificam o desenvolvimento de esquemas de banco de dados e a criação de consultas SQL complexas.

5. **Administração de Segurança**: Facilita a administração de permissões e segurança no banco de dados, permitindo aos administradores gerenciar logons, usuários, papéis e permissões de forma centralizada.

6. **Automação de Tarefas**: Oferece suporte à automação de tarefas administrativas por meio de scripts T-SQL, procedimentos armazenados, jobs do SQL Server Agent e integração com PowerShell, permitindo agendar e executar rotinas de manutenção, backups e outras tarefas programadas.

7. **Monitoramento de Desempenho**: Inclui ferramentas para monitoramento e otimização de desempenho, como o Monitor de Atividade e Relatórios de Desempenho, que ajudam a identificar gargalos de desempenho e ajustar configurações para melhorar a eficiência do banco de dados.

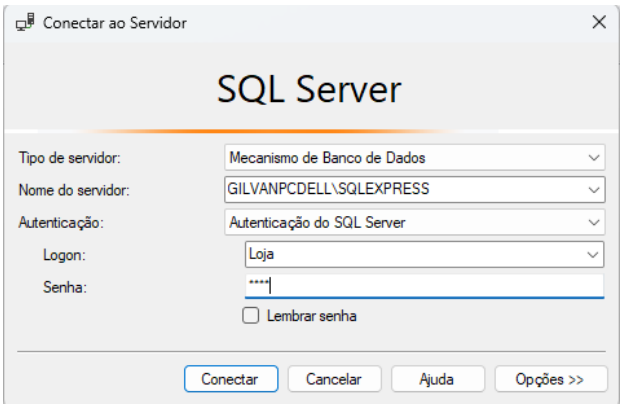
8. **Integração com Ferramentas de BI**: Possibilita a integração com ferramentas de Business Intelligence (BI), como o SQL Server Reporting Services (SSRS) e o SQL Server Analysis Services (SSAS), facilitando a criação e gerenciamento de relatórios e análises.

Esses recursos combinados ajudam os administradores de banco de dados e desenvolvedores a serem mais produtivos, permitindo-lhes gerenciar, desenvolver e manter bancos de dados SQL Server de maneira eficiente e eficaz através de uma interface unificada e poderosa.

2º Procedimento | Alimentando a Base de dados

1. Utilizar o SQL Server Management Studio para alimentar as tabelas com dados básicos do sistema:

   a. Logar como usuário **loja**, senha **loja.**



   b. Utilizar o editor de SQL para incluir dados na tabela de usuários, de forma a obter um conjunto como o apresentado a seguir:

```
INSERT INTO Usuario(loginName,senha) VALUES ('op1','op1'), ('op2','op2');

SELECT TOP (1000) [idUsuario] ,[loginName] ,[senha] FROM [Loja].[dbo].[Usuario]
```

100 %

Resultados | Mensagens

	idUsuario	loginName	senha
1	1	op1	op1
2	2	op2	op2

   c. Inserir alguns produtos na base de dados, obtendo um conjunto como o que é apresentado a seguir:

```
INSERT INTO Produto(nome,quantidade,precoVenda) VALUES ('Banana',100,'5.00'), ('Laranja',500,'2.00'), ('Manga',800,'4.00');

SELECT TOP (1000) [idProduto],[nome],[quantidade],[precoVenda] FROM [Loja].[dbo].[Produto]
```

100 %

Resultados | Mensagens

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

2. Criar pessoas físicas e jurídicas na base de dados:

a. Obter o próximo id de pessoa a partir da sequence.

```
CREATE SEQUENCE orderPessoa
AS INT
START WITH 1
INCREMENT BY 1;
```

b. Incluir na tabela pessoa os dados comuns

```
INSERT INTO Pessoa(idPessoa,nome,endereco,cidade,estado,telefone,email) VALUES
(NEXT VALUE FOR orderPessoa, 'Gilvan Junior','Rua Violeta Areosa, 102','Manaus','AM','3238-1408','gilvanx10@gmail.com'),
(NEXT VALUE FOR orderPessoa, 'Adriana Reis Mota','Rua Dias Candido, 45','Normandia','RO','3234-5588','adia.arm@gmail.com'),
(NEXT VALUE FOR orderPessoa, 'Ramona Morgana','Rua Deodato de Miranda Leão, 01','Manaus','AM','9988-4407','ramona@yahoo.com');


SELECT TOP (1000) [idPessoa]
      ,[nome]
      ,[endereco]
      ,[cidade]
      ,[estado]
      ,[telefone]
      ,[email]
  FROM [Loja].[dbo].[Pessoa]
```

100 %

Resultados | Mensagens

	idPessoa	nome	endereco	cidade	estado	telefone	email
1	1	Gilvan Junior	Rua Violeta Areosa, 102	Manaus	AM	3238-1408	gilvanx10@gmail.com
2	2	Adriana Reis Mota	Rua Dias Candido, 45	Normandia	RO	3234-5588	adia.arm@gmail.com
3	3	Ramona Morgana	Rua Deodato de Miranda Leão, 01	Manaus	AM	9988-4407	ramona@yahoo.com

c. Incluir em pessoa física o CPF...,

```
INSERT INTO PessoaFisica(FK_Pessoa_idPessoa,cpf) VALUES (1,'76023778003'), (2,'21237386039'), (3,'21553800036');

SELECT TOP (1000) [FK_Pessoa_idPessoa],[cpf] FROM [Loja].[dbo].[PessoaFisica]
```

100 %

Resultados | Mensagens

	FK_Pessoa_idPessoa	cpf
1	1	76023778003
2	2	21237386039
3	3	21553800036

...efetuando o relacionamento com pessoa.

```
SELECT p.*, pf.cpf FROM Pessoa p INNER JOIN PessoaFisica pf ON p.idPessoa = pf.FK_Pessoa_idPessoa;
```

100 %

Resultados | Mensagens

	idPessoa	nome	endereco	cidade	estado	telefone	email	cpf
1	1	Gilvan Junior	Rua Violeta Areosa, 102	Manaus	AM	3238-1408	gilvanx10@gmail.com	76023778003
2	2	Adriana Reis Mota	Rua Dias Candido, 45	Normandia	RO	3234-5588	adia.arm@gmail.com	21237386039
3	3	Ramona Morgana	Rua Deodato de Miranda Leão, 01	Manaus	AM	9988-4407	ramona@yahoo.com	21553800036

d. Incluir em pessoa jurídica o CNPJ...,

```
INSERT INTO PessoaJuridica(FK_Pessoa_idPessoa,cnpj) VALUES
(4,'72948678000101'),
(5,'76725807000153'),
(6,'03451259000147'),
(7,'76865641000170');

SELECT TOP (1000) [FK_Pessoa_idPessoa],[cnpj] FROM [Loja].[dbo].[PessoaJuridica]
```

100 %

Resultados | Mensagens

	FK_Pessoa_idPessoa	cnpj
1	4	72948678000101
2	5	76725807000153
3	6	03451259000147
4	7	76865641000170

...efetuando o relacionando com pessoa.

```
SELECT p.*, pj.cnpj
FROM Pessoa p
INNER JOIN PessoaJuridica pj ON p.idPessoa = pj.FK_Pessoa_idPessoa;
```

100 %

Resultados | Mensagens

	idPessoa	nome	endereco	cidade	estado	telefone	email	cnpj
1	4	Restaurante Dias	Av. Epaminondas, 105	Manaus	AM	5534-4582	rastdias@restaurantedias.com.br	72948678000101
2	5	Panificadora Servepam	BL C CJ. Algusto Monte Negro, 987	Campinas	SP	0954-7821	servepam@yahoo.com.br	76725807000153
3	6	Eletrotec HLM	RUA 21 , 458	Araras	SP	3234-9988	eletrotechlm@gmail.com.br	03451259000147
4	7	Auto Escola Rubens	Beco Agro, 02	Guarulhos	SP	3954-8592	bibike@bibiki.com.br	76865641000170

3. Criar algumas movimentações na base de dados, obtendo um conjunto como o que é apresentado a seguir, onde E representa Entrada e S representa Saída.

```
INSERT INTO Movimento(FK_Usuario_idUsuario,FK_Pessoa_idPessoa,FK_Produto_idProduto,quantidade,tipo,precoUnitario)
VALUES
(1,1,1,10,'E',1.00),
(2,2,2,20,'S',2.00),
(1,3,3,30,'E',3.00);

SELECT TOP (1000) [idMovimento],[FK_Usuario_idUsuario],[FK_Pessoa_idPessoa],
[FK_Produto_idProduto],[quantidade],[tipo],[precoUnitario]
FROM [Loja].[dbo].[Movimento]
```

100 %

Resultados | Mensagens

	idMovimento	FK_Usuario_idUsuario	FK_Pessoa_idPessoa	FK_Produto_idProduto	quantidade	tipo	precoUnitario
1	1	1	1	1	10	E	1.00
2	2	2	2	2	20	S	2.00
3	3	1	3	3	30	E	3.00

4. Efetuar as seguintes consultas sobre os dados inseridos:

   a. Dados completos de pessoas físicas.

```
SELECT p.*, pf.cpf
FROM Pessoa p
INNER JOIN PessoaFisica pf ON p.idPessoa = pf.FK_Pessoa_idPessoa;
```

100 %

Resultados | Mensagens

	idPessoa	nome	endereco	cidade	estado	telefone	email	cpf
1	1	Gilvan Junior	Rua Violeta Areosa, 102	Manaus	AM	3238-1408	gilvanx10@gmail.com	76023778003
2	2	Adriana Reis Mota	Rua Dias Candido, 45	Normandia	RO	3234-5588	adia.arm@gmail.com	21237386039
3	3	Ramona Morgana	Rua Deodato de Miranda Leão, 01	Manaus	AM	9988-4407	ramona@yahoo.com	21553800036

   b. Dados completos de pessoas jurídicas.

```
SELECT p.*, pj.cnpj
FROM Pessoa p
INNER JOIN PessoaJuridica pj ON p.idPessoa = pj.FK_Pessoa_idPessoa;
```

100 %

Resultados | Mensagens

	idPessoa	nome	endereco	cidade	estado	telefone	email	cnpj
1	4	Restaurante Dias	Av. Epaminondas, 105	Manaus	AM	5534-4582	rastdias@restaurantedias.com.br	72948678000101
2	5	Panificadora Servepam	BL C CJ. Algusto Monte Negro, 987	Campinas	SP	0954-7821	servepam@yahoo.com.br	76725807000153
3	6	Eletrotec HLM	RUA 21 , 458	Araras	SP	3234-9988	eletrotechlm@gmail.com.br	03451259000147
4	7	Auto Escola Rubens	Beco Agro, 02	Guarulhos	SP	3954-8592	bibike@bibiki.com.br	76865641000170

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

```
SELECT m.*, p.nome as fornecedor, pr.nome as Produto, m.quantidade, m.precoUnitario,
(m.quantidade * m.precoUnitario) as total FROM Movimento m INNER JOIN Pessoa p ON p.idPessoa = m.FK_Pessoa_idPessoa
INNER JOIN Produto pr ON pr.idProduto = m.FK_Produto_idProduto WHERE m.tipo = 'E';
```

00 %

Resultados | Mensagens

	idMovimento	FK_Usuario_idUsuario	FK_Pessoa_idPessoa	FK_Produto_idProduto	quantidade	tipo	precoUnitario	fornecedor	Produto	quantidade	precoUnitario	total
1	1	1	1	1	10	E	1.00	Gilvan Junior	Banana	10	1.00	10.00
2	3	1	3	3	30	E	3.00	Ramona Morgana	Manga	30	3.00	90.00

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

```
SELECT mov.*, p.nome as comprador, pr.nome as Produto, mov.quantidade, mov.precoUnitario, (mov.quantidade * mov.precoUnitario) as total
FROM Movimento mov INNER JOIN Pessoa p ON mov.FK_Pessoa_idPessoa = p.idPessoa
INNER JOIN Produto pr ON mov.FK_Produto_idProduto = pr.idProduto
WHERE mov.tipo = 'S';
```

00 %

Resultados | Mensagens

	idMovimento	FK_Usuario_idUsuario	FK_Pessoa_idPessoa	FK_Produto_idProduto	quantidade	tipo	precoUnitario	comprador	Produto	quantidade	precoUnitario	total
1	2	2	2	2	20	S	2.00	Adriana Reis Mota	Laranja	20	2.00	40.00

e. Valor total das entradas agrupadas por produto.

```
SELECT pr.nome, SUM(m.quantidade * m.precoUnitario) as compras
FROM Movimento m
INNER JOIN Produto pr ON m.FK_Produto_idProduto = pr.idProduto
WHERE m.tipo = 'E'
GROUP BY pr.nome;
```

00 %

Resultados | Mensagens

	nome	compras
1	Banana	10.00
2	Manga	90.00

f. Valor total das saídas agrupadas por produto.

```
SELECT pr.nome, SUM(m.quantidade * m.precoUnitario) as vendas
FROM Movimento m
INNER JOIN Produto pr ON m.FK_Produto_idProduto = pr.idProduto
WHERE m.tipo = 'S'
GROUP BY pr.nome;
```

100 %

Resultados | Mensagens

	nome	vendas
1	Laranja	40.00

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

```
SELECT u.*
FROM Usuario u
LEFT JOIN Movimento m ON u.idUsuario = m.FK_Usuario_idUsuario AND m.tipo = 'E'
WHERE m.idMovimento IS NULL;
```

100 %

Resultados | Mensagens

	idUsuario	loginName	senha
1	2	op2	op2

h. Valor total de entrada, agrupado por operador.

```
SELECT u.loginName, SUM(m.precoUnitario * m.quantidade) as compras
FROM Movimento m
INNER JOIN Usuario u ON m.FK_Usuario_idUsuario = u.idUsuario
WHERE m.tipo = 'E'
GROUP BY u.loginName;
```

100 %

Resultados | Mensagens

	loginName	compras
1	op1	100.00

i. Valor total de saída, agrupado por operador.

```
SELECT u.loginName, SUM(m.precoUnitario * m.quantidade) as vendas
FROM Movimento m
INNER JOIN Usuario u ON m.FK_Usuario_idUsuario = u.idUsuario
WHERE m.tipo = 'S'
GROUP BY u.loginName;
```

.00 %

Resultados | Mensagens

	loginName	vendas
1	op2	40.00

j. Valor médio de venda por produto, utilizando média ponderada.

```
SELECT pr.nome, SUM(mov.precoUnitario * mov.quantidade) / SUM(mov.quantidade) as media
FROM Movimento mov
INNER JOIN Produto pr ON mov.FK_Produto_idProduto = pr.idProduto
WHERE mov.tipo = 'S'
GROUP BY pr.nome;
```

.00 %

Resultados | Mensagens

	nome	media
1	Laranja	2.000000

Análise e Conclusão do 2º Procedimento:

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

As diferenças principais no uso de SEQUENCE e IDENTITY em SQL estão relacionadas ao seu funcionamento, sintaxe e suporte em diferentes sistemas de gerenciamento de banco de dados (SGBDs). Aqui estão os pontos chave:

***SEQUENCE**: Oferece mais flexibilidade porque pode ser compartilhada por várias tabelas e permite mais controle sobre como os valores são obtidos e utilizados.*

***IDENTITY**: É mais simples de implementar e usar em comparação com sequence, especialmente quando se trata de tabelas simples que precisam de uma chave primária auto-incremental.*

As escolha entre SEQUENCE e IDENTITY depende das necessidades específicas do banco de dados e das características de implementação desejadas. Por exemplo, se você precisa de valores únicos que possam ser compartilhados entre várias tabelas, SEQUENCE é mais adequada. Se você precisa de uma chave primária auto-incremental simples em uma única tabela, IDENTITY pode ser mais conveniente.
Em resumo, ambas as opções são úteis para a geração automática de valores únicos em bancos de dados relacionais, mas cada uma tem suas próprias características e casos de uso específicos.

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

As chaves estrangeiras (foreign keys) desempenham um papel crucial na garantia da consistência e integridade dos dados em um banco de dados relacional. Aqui estão as principais razões pelas quais as chaves estrangeiras são importantes:

a. *Manutenção da Integridade Referencial:*
b. *Evita Inconsistências de Dados:*
c. *Facilita a Manutenção e Gestão de Dados:*
d. *Suporte a Operações Transacionais:*
e. *Documentação e Entendimento do Modelo de Dados:*

Em resumo, as chaves estrangeiras são fundamentais para garantir que as relações entre dados em um banco de dados relacional sejam consistentes, corretas e seguras. Elas ajudam a evitar erros de integridade referencial, simplificam operações de manutenção e consultas complexas, e contribuem para a confiabilidade e eficiência do sistema de gerenciamento de banco de dados como um todo.

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

No contexto de bancos de dados relacionais, os operadores podem ser classificados em duas principais abordagens teóricas: álgebra relacional e cálculo relacional. Aqui estão os operadores típicos de cada uma dessas abordagens:

Álgebra Relacional:

1. ***Seleção (Selection):***
   - *Operador representado por σ.*
   - *Seleciona linhas da tabela que satisfazem uma condição específica.*
2. ***Projeção (Projection):***
   - *Operador representado por π.*
   - *Seleciona colunas específicas da tabela.*
3. ***União (Union):***
   - *Operador representado por ∪.*
   - *Combina duas relações (tabelas) que têm o mesmo conjunto de colunas.*
4. ***Interseção (Intersection):***
   - *Operador representado por ∩.*
   - *Retorna as linhas que estão presentes em ambas as relações.*
5. ***Diferença (Difference):***
   - *Operador representado por - (ou).*
   - *Retorna as linhas que estão na primeira relação, mas não na segunda.*
6. ***Produto Cartesiano (Cartesian Product):***
   - *Operador representado por ×.*
   - *Combina cada linha de uma relação com cada linha de outra relação.*
7. ***Junção (Join):***
   - *Operador que combina colunas de duas ou mais tabelas com base em uma condição relacionada.*

Cálculo Relacional:

O cálculo relacional não possui operadores tão explicitamente definidos quanto a álgebra relacional. Em vez disso, ele utiliza expressões e predicados para especificar o que deve ser recuperado da base de dados. Alguns conceitos no cálculo relacional incluem:

1. ***Cálculo de Tupla (Tuple Calculus):***
   - *Especifica as tuplas (linhas) que satisfazem uma determinada condição.*
   - *Utiliza variáveis para descrever as condições e expressões para filtrar e selecionar as tuplas.*
2. ***Cálculo de Domínio (Domain Calculus):***
   - *Especifica os valores de atributo (coluna) que satisfazem uma determinada condição.*
   - *Também utiliza variáveis e expressões para definir as condições de seleção.*

O cálculo relacional, por outro lado, define uma linguagem mais abrangente para descrever e manipular dados em um banco de dados relacional. Ele utiliza fórmulas matemáticas para expressar consultas complexas, incluindo quantificadores e variáveis. No SQL, alguns operadores são influenciados pelo cálculo relacional, mas a implementação completa não se limita a ele.

- *Agregação: Funções como COUNT, SUM, AVG, MIN e MAX são utilizadas para resumir dados em grupos de tuplas, retornando valores agregados. No SQL, essas funções são geralmente usadas em conjunto com a cláusula GROUP BY.*
- *Aninhamento de Consultas: O SQL permite aninhar consultas umas dentro das outras, utilizando subconsultas, o que possibilita realizar operações mais complexas e combinações de dados de diferentes relações.*

Nem todos os operadores da álgebra relacional possuem uma representação direta no SQL. Algumas operações podem ser implementadas através de combinações de outros operadores ou utilizando recursos mais avançados da linguagem.
O cálculo relacional fornece uma base teórica mais poderosa para a manipulação de dados, mas o SQL oferece uma sintaxe mais prática e intuitiva para a maioria dos usuários.

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

O agrupamento em consultas SQL é feito usando a cláusula GROUP BY. Esta cláusula permite agrupar linhas que possuem valores iguais em uma ou mais colunas especificadas, geralmente para aplicar funções de agregação como SUM, COUNT, AVG, MAX, MIN sobre grupos de dados. Sintaxe Básica do "GROUP BY":

```
SELECT coluna1, coluna2, ..., funcao_agregacao(coluna)
FROM tabela
GROUP BY coluna1, coluna2, ...
```

Requisitos Obrigatórios para o GROUP BY:

Para utilizar o GROUP BY de forma correta e eficaz, é fundamental seguir dois requisitos:

1. ***Especificar Colunas de Agrupamento:***
   - *Ao usar GROUP BY, você precisa especificar quais colunas serão usadas para agrupar os dados. Isso significa que todas as colunas na lista SELECT que não são funções de agregação devem estar presentes na lista GROUP BY.*
   - *Por exemplo, se você selecionar coluna1, coluna2 e aplicar SUM(coluna3), então coluna1 e coluna2 devem estar presentes na cláusula GROUP BY.*
2. ***Restrições na Seleção de Colunas:***
   - *Quando você usa funções de agregação (SUM, COUNT, AVG, etc.) em uma consulta com GROUP BY, geralmente não pode selecionar colunas individuais que não estão incluídas na cláusula GROUP BY (a menos que elas estejam dentro de uma função de agregação).*

- *Por exemplo, se você agrupa por coluna1 e coluna2, e deseja contar quantos registros existem em cada grupo, você pode fazer COUNT(*) ou COUNT(coluna_qualquer), mas não pode selecionar coluna3 diretamente a menos que seja uma função de agregação.*

Exemplo Prático:

Considere uma tabela pedidos com as colunas id_pedido, id_cliente, data_pedido e valor_pedido. Se quisermos calcular o total de pedidos por cliente:

```
SELECT id_cliente, COUNT(*) AS total_pedidos
FROM pedidos
GROUP BY id_cliente;
```

Neste exemplo:

- *Estamos agrupando os pedidos pelo id_cliente.*
- *A função de agregação COUNT(*) conta quantos registros (pedidos) existem para cada id_cliente.*
- *id_cliente é especificado na cláusula GROUP BY, pois estamos agrupando pelos valores únicos dessa coluna.*

Portanto, o requisito obrigatório ao usar GROUP BY em consultas SQL é especificar quais colunas estão sendo usadas para agrupar os dados na cláusula GROUP BY. Isso garante que a operação de agrupamento seja feita de maneira correta e que as funções de agregação sejam aplicadas aos grupos de dados desejados.