‘

Universida de Estácio de Sá

DESENVOLVIMENTO FULL STACK

• Disciplina: RPG0014 - Iniciando o caminho pelo Java

• Semestre Letivo: 2024.3

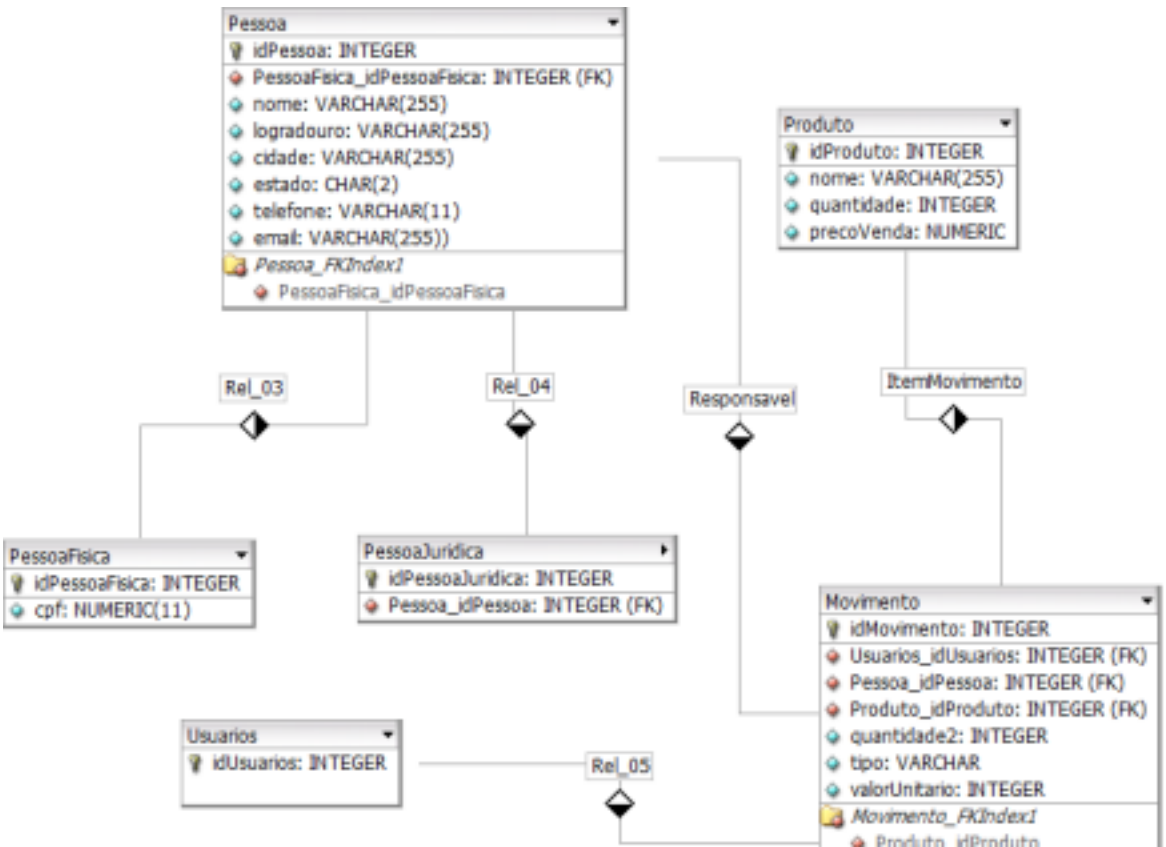
DARCI RODRIGUES FALCÃO NETO

Missão Prática | Nível 1 | Mundo

Objetivo 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.**

Bom de modo geral eu juntei os dois códigos do procedimento, e resolvi trazer em apenas 1 código tudo junto. segue o mesmo abaixo.


Pessoa
idPessoa: INTEGER
PessoaFisica_idPessoaFisica: INTEGER (FK)
nome: VARCHAR(255)
logradouro: VARCHAR(255)
cidade: VARCHAR(255)
estado: CHAR(2)
telefone: VARCHAR(11)
email: VARCHAR(255))
Pessoa_FKIndex1
PessoaFisica_idPessoaFisica
Produto
idProduto: INTEGER
nome: VARCHAR(255)
quantidade: INTEGER
precoVenda: NUMERIC
Rel_03
Rel_04
Responsavel
ItemMovimento
PessoaFisica
idPessoaFisica: INTEGER
cpf: NUMERIC(11)
PessoaJuridica
idPessoaJuridica: INTEGER
Pessoa_idPessoa: INTEGER (FK)
Movimento
idMovimento: INTEGER
Usuarios_idUsuarios: INTEGER (FK)
Pessoa_idPessoa: INTEGER (FK)
Produto_idProduto: INTEGER (FK)
quantidade2: INTEGER
tipo: VARCHAR
valorUnitario: INTEGER
Movimento_FKIndex1
Usuarios
idUsuarios: INTEGER
Rel_05

```
-- Criar o banco de dados SistemaLoja
CREATE DATABASE SistemaLoja;

-- Usar o banco de dados recém-criado
USE SistemaLoja;

-- Criar a tabela de Usuários
CREATE TABLE Usuarios (
    IDUsuario INT PRIMARY KEY,
    Nome VARCHAR(255),
    Senha VARCHAR(255)
);

-- Criar a tabela de Produtos
CREATE TABLE Produtos (
    IDProduto INT PRIMARY KEY,
    Nome VARCHAR(255),
```

```
    Quantidade INT,
    PrecoVenda DECIMAL(10, 2)
);

-- Criar a tabela de Pessoas
CREATE TABLE Pessoas (
    IDPessoa INT PRIMARY KEY,
    Nome VARCHAR(255),
    Endereco VARCHAR(255),
    Cidade VARCHAR(255),
    Estado VARCHAR(255),
    Telefone VARCHAR(255),
    Email VARCHAR(255)
);
```

```
-- Criar a tabela de PessoasFisica
CREATE TABLE PessoasFisica (
    IDPessoa INT PRIMARY KEY,
    CPF VARCHAR(14) UNIQUE
);

-- Criar a tabela de PessoasJuridica
CREATE TABLE PessoasJuridica (
    IDPessoa INT PRIMARY KEY,
    CNPJ VARCHAR(18) UNIQUE
);

-- Criar a tabela de MovimentoCompra (Entrada)
CREATE TABLE MovimentoCompra (
    IDMovimento INT PRIMARY KEY,
    IDProduto INT,
    IDPessoaJuridica INT,
    Quantidade INT,
    PrecoUnitario DECIMAL(10, 2),
    FOREIGN KEY (IDProduto) REFERENCES Produtos(IDProduto), FOREIGN
KEY (IDPessoaJuridica) REFERENCES PessoasJuridicas(IDPessoa) );

-- Criar a tabela de MovimentoVenda (Saída)
CREATE TABLE MovimentoVenda (
    IDMovimento INT PRIMARY KEY,
    IDProduto INT,
    IDPessoaFisica INT,
    Quantidade INT,
    FOREIGN KEY (IDProduto) REFERENCES Produtos(IDProduto),
    FOREIGN KEY (IDPessoaFisica) REFERENCES
    PessoasFisicas(IDPessoa)
```

```
);

-- Criar uma sequence para gerar IDs de Pessoa
CREATE SEQUENCE IdentificadorPessoa START WITH 1 INCREMENT BY 1;

-- Criar uma sequence para gerar IDs de Usuário
CREATE SEQUENCE IdentificadorUsuario START WITH 1 INCREMENT BY 1;

-- Criar uma sequence para gerar IDs de Produto
CREATE SEQUENCE IdentificadorProduto START WITH 1 INCREMENT BY 1;

-- Criar uma sequence para gerar IDs de MovimentoCompra
CREATE SEQUENCE IdentificadorMovimentoCompra START WITH 1 INCREMENT BY
1;
```

-- Criar uma sequence para gerar IDs de MovimentoVenda
CREATE SEQUENCE IdentificadorMovimentoVenda START WITH 1 INCREMENT BY 1;

códigos para inserir dados:

1. Inserir dados na tabela de Usuários:

```sql
INSERT INTO Usuarios (IDUsuario, Nome, Senha)
VALUES
    (1, 'bruno', '123'),
    (2, 'lorenzo', '123');
```

2. Inserir dados na tabela de Produtos:

```sql
INSERT INTO Produtos (IDProduto, Nome, Quantidade, PrecoVenda)
VALUES
    (1, 'Banana', 100, 10.50),
    (2, 'Abacate', 50, 20.75),
    (3, 'Melancia', 200, 5.25);
```

3. Inserir dados nas tabelas de Pessoas e Pessoas Físicas:

```sql
-- Inserir dados na tabela de Pessoas (Pessoa Física)
```

```
DECLARE @ProximoID INT;
SET @ProximoID = NEXT VALUE FOR IdentificadorPessoa;

INSERT INTO Pessoas (IDPessoa, Nome, Endereco,
Telefone) VALUES
    (@ProximoID, 'Fulano', 'Rua A, 123', '123-456-7890');

INSERT INTO PessoaFisica (IDPessoa, CPF)
VALUES
    (@ProximoID, '123.456.789-00');
```

4. Inserir dados nas tabelas de Pessoas e Pessoas Jurídicas:

```sql
```

```
-- Inserir dados na tabela de Pessoas (Pessoa Jurídica)
SET @ProximoID = NEXT VALUE FOR IdentificadorPessoa;

INSERT INTO Pessoas (IDPessoa, Nome, Endereco,
Telefone) VALUES
    (@ProximoID, 'Empresa ABC Ltda', 'Av. B, 456', '987-654-3210');

INSERT INTO PessoaJuridica (IDPessoa, CNPJ)
VALUES
    (@ProximoID, '12.345.678/0001-23');
```

5. Inserir dados na tabela de MovimentoVenda (Saída):

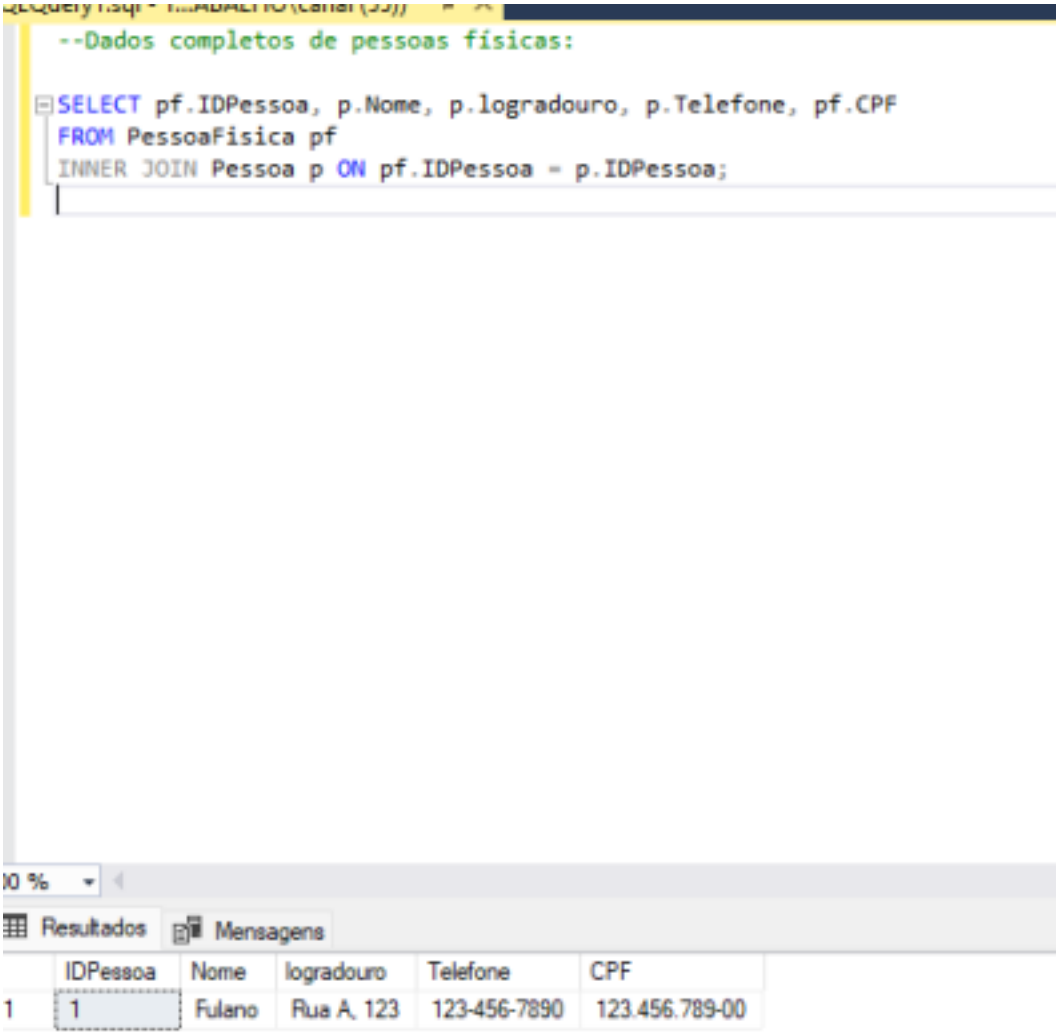
```sql
INSERT INTO MovimentoVenda (IDMovimento, IDProduto,
IDPessoaFisica, Quantidade)
VALUES
    (4, 1, 1, 10), -- Substitua 1 pelo ID da Pessoa Física
    (5, 2, 1, 5), -- Substitua 1 pelo ID da Pessoa Física
    (6, 3, 1, 20); -- Substitua 1 pelo ID da Pessoa Física


```

códigos para consultar dados

Dados completos de pessoas físicas:

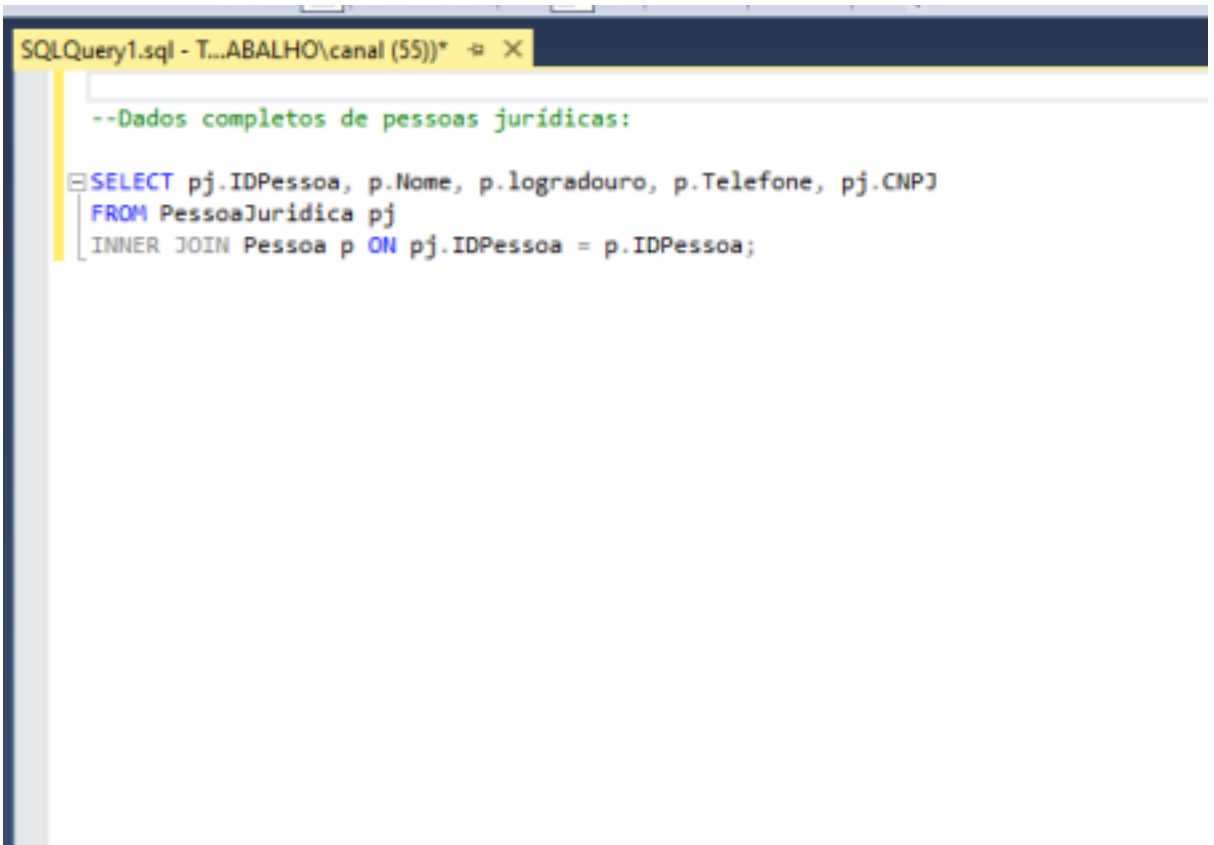
```
SELECT pf.IDPessoa, p.Nome, p.logradouro, p.Telefone, pf.CPF
```

```
FROM PessoaFisica pf
INNER JOIN Pessoa p ON pf.IDPessoa = p.IDPessoa;
```



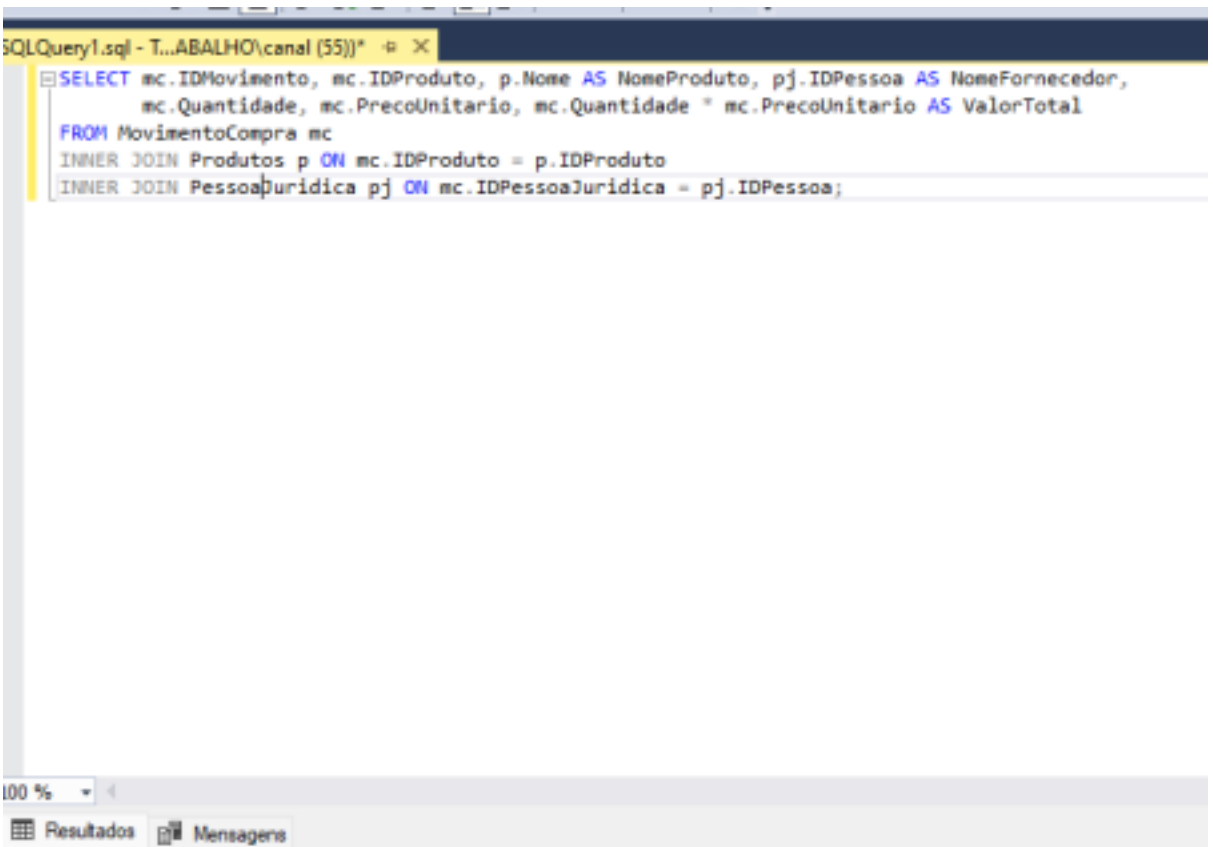
Dados completos de pessoas jurídicas:

SELECT pj.IDPessoa, p.Nome, p.logradouro, p.Telefone, pj.CNPJ
FROM PessoaJuridica pj
INNER JOIN Pessoa p ON pj.IDPessoa = p.IDPessoa;


SQLQuery1.sql - T...ABALHO\canal (55))*
--Dados completos de pessoas jurídicas:
SELECT pj.IDPessoa, p.Nome, p.logradouro, p.Telefone, pj.CNPJ
FROM PessoaJuridica pj
INNER JOIN Pessoa p ON pj.IDPessoa = p.IDPessoa;

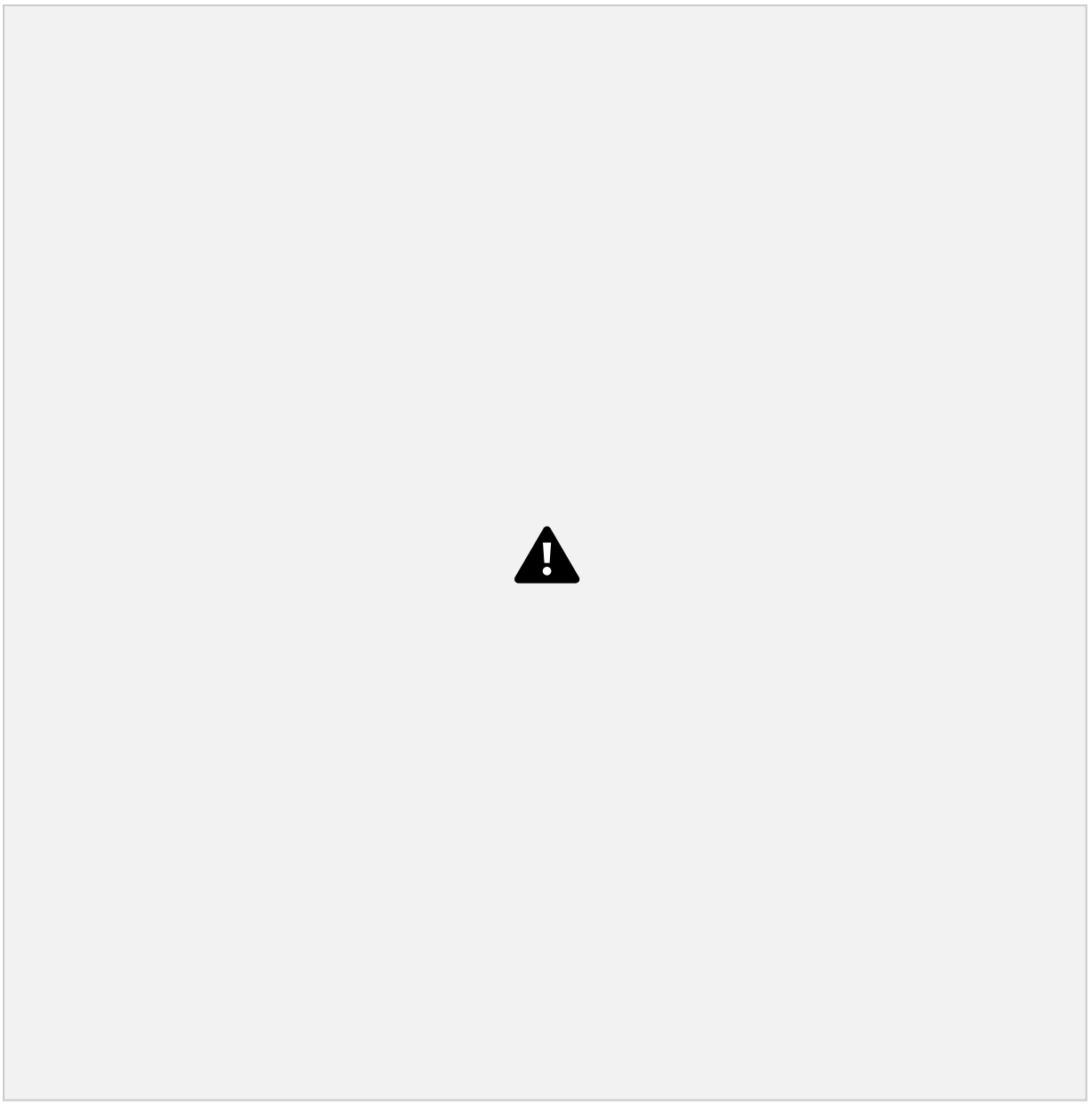
Movimentações de entrada (compra) com detalhes:

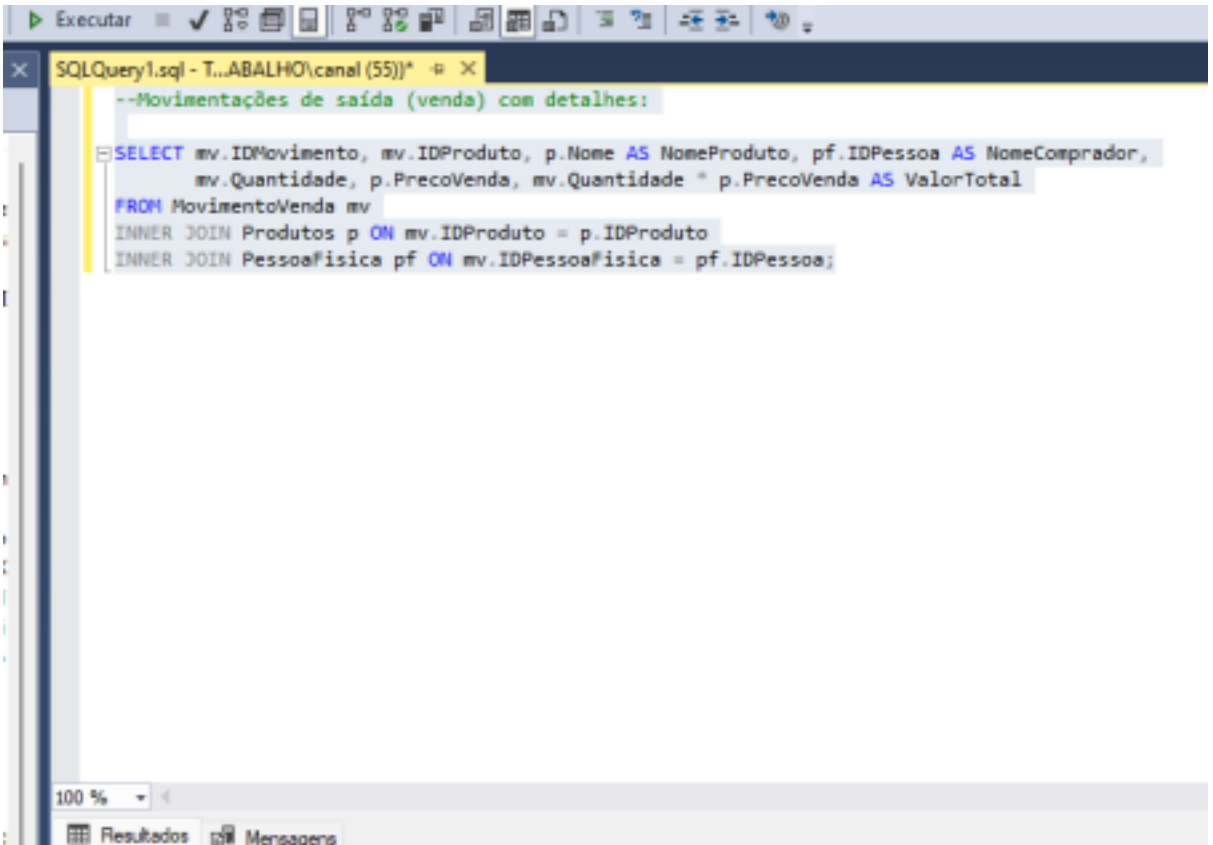
```
SELECT mc.IDMovimento, mc.IDProduto, p.Nome AS NomeProduto, pj.IDPessoa AS
NomeFornecedor,
    mc.Quantidade, mc.PrecoUnitario, mc.Quantidade * mc.PrecoUnitario AS
ValorTotal FROM MovimentoCompra mc
INNER JOIN Produtos p ON mc.IDProduto = p.IDProduto
INNER JOIN PessoaJuridica pj ON mc.IDPessoaJuridica = pj.IDPessoa;
```


SQLQuery1.sql - T...ABALHO\canal (55))*
SELECT mc.IDMovimento, mc.IDProduto, p.Nome AS NomeProduto, pj.IDPessoa AS NomeFornecedor,
mc.Quantidade, mc.PrecoUnitario, mc.Quantidade * mc.PrecoUnitario AS ValorTotal
FROM MovimentoCompra mc
INNER JOIN Produtos p ON mc.IDProduto = p.IDProduto
INNER JOIN PessoaJuridica pj ON mc.IDPessoaJuridica = pj.IDPessoa;
100 %
Resultados
Mensagens

--Movimentações de saída (venda) com detalhes:

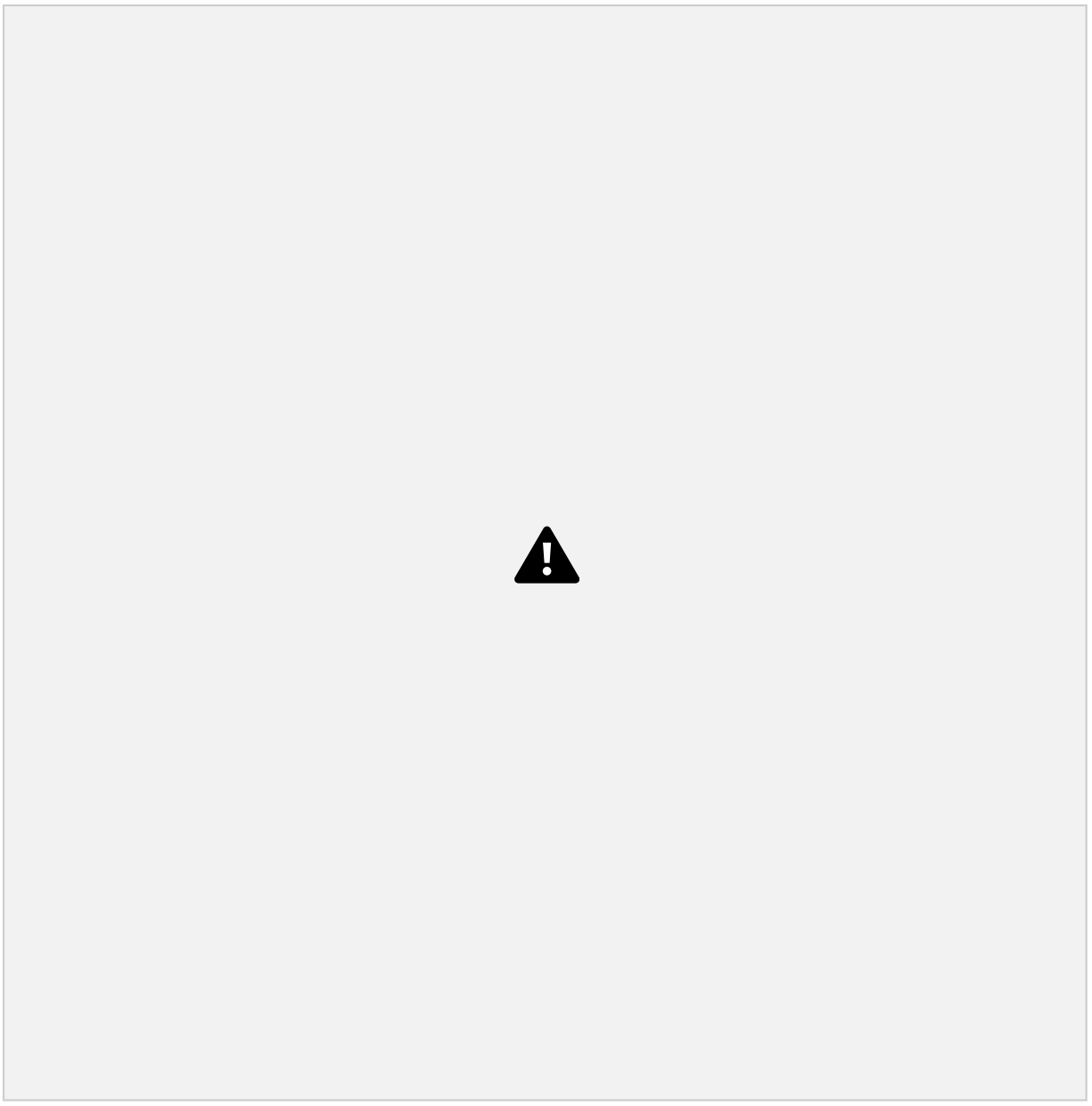
```
SELECT mv.IDMovimento, mv.IDProduto, p.Nome AS NomeProduto, pf.IDPessoa AS
NomeComprador,
    mv.Quantidade, p.PrecoVenda, mv.Quantidade * p.PrecoVenda AS
ValorTotal FROM MovimentoVenda mv
INNER JOIN Produtos p ON mv.IDProduto = p.IDProduto
INNER JOIN PessoaFisica pf ON mv.IDPessoaFisica = pf.IDPessoa;
```

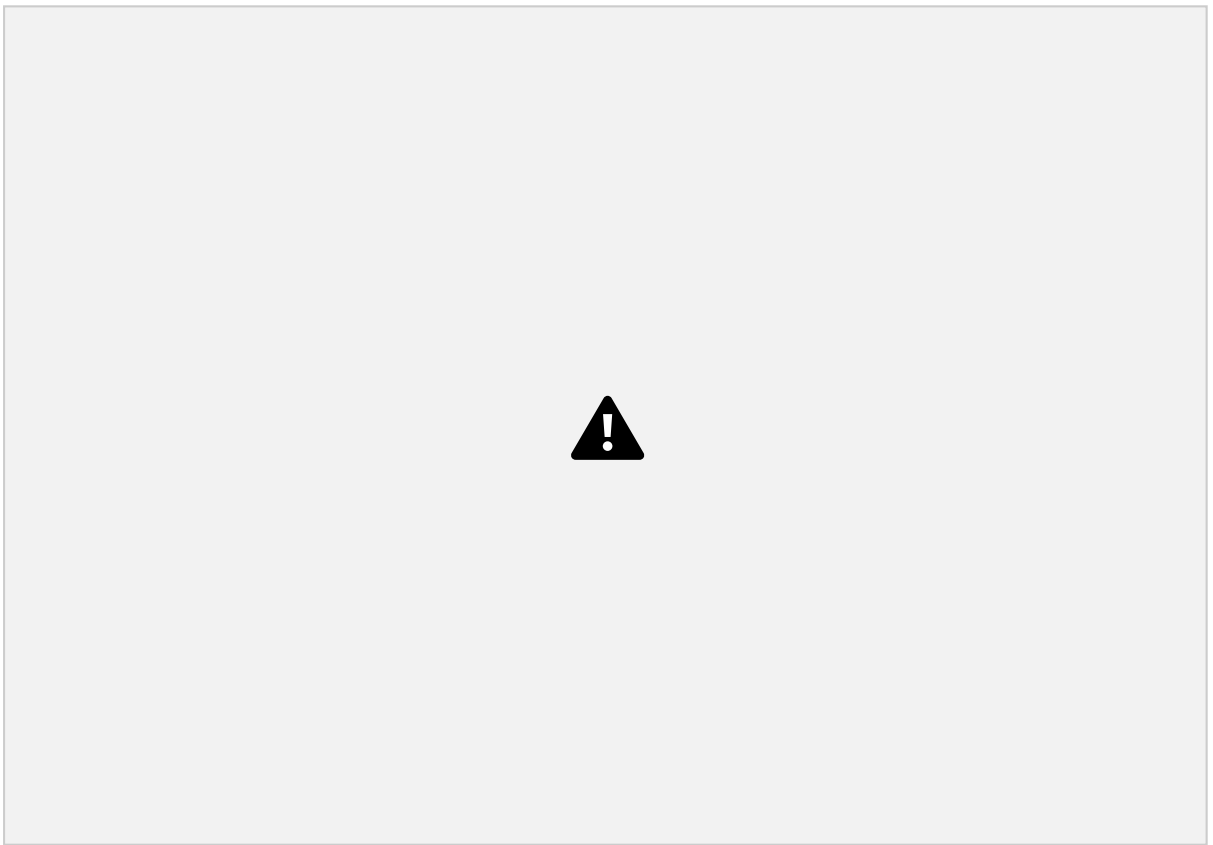



Executar
SQLQuery1.sql - T...ABALHO\canal (55))*
--Movimentações de saída (venda) com detalhes:
SELECT mv.IDMovimento, mv.IDProduto, p.Nome AS NomeProduto, pf.IDPessoa AS NomeComprador,
mv.Quantidade, p.PrecoVenda, mv.Quantidade * p.PrecoVenda AS ValorTotal
FROM MovimentoVenda mv
INNER JOIN Produtos p ON mv.IDProduto = p.IDProduto
INNER JOIN Pessoafisica pf ON mv.IDPessoafisica = pf.IDPessoa;
100 %
Resultados
Mensagens

--Valor total das entradas agrupadas por produto:

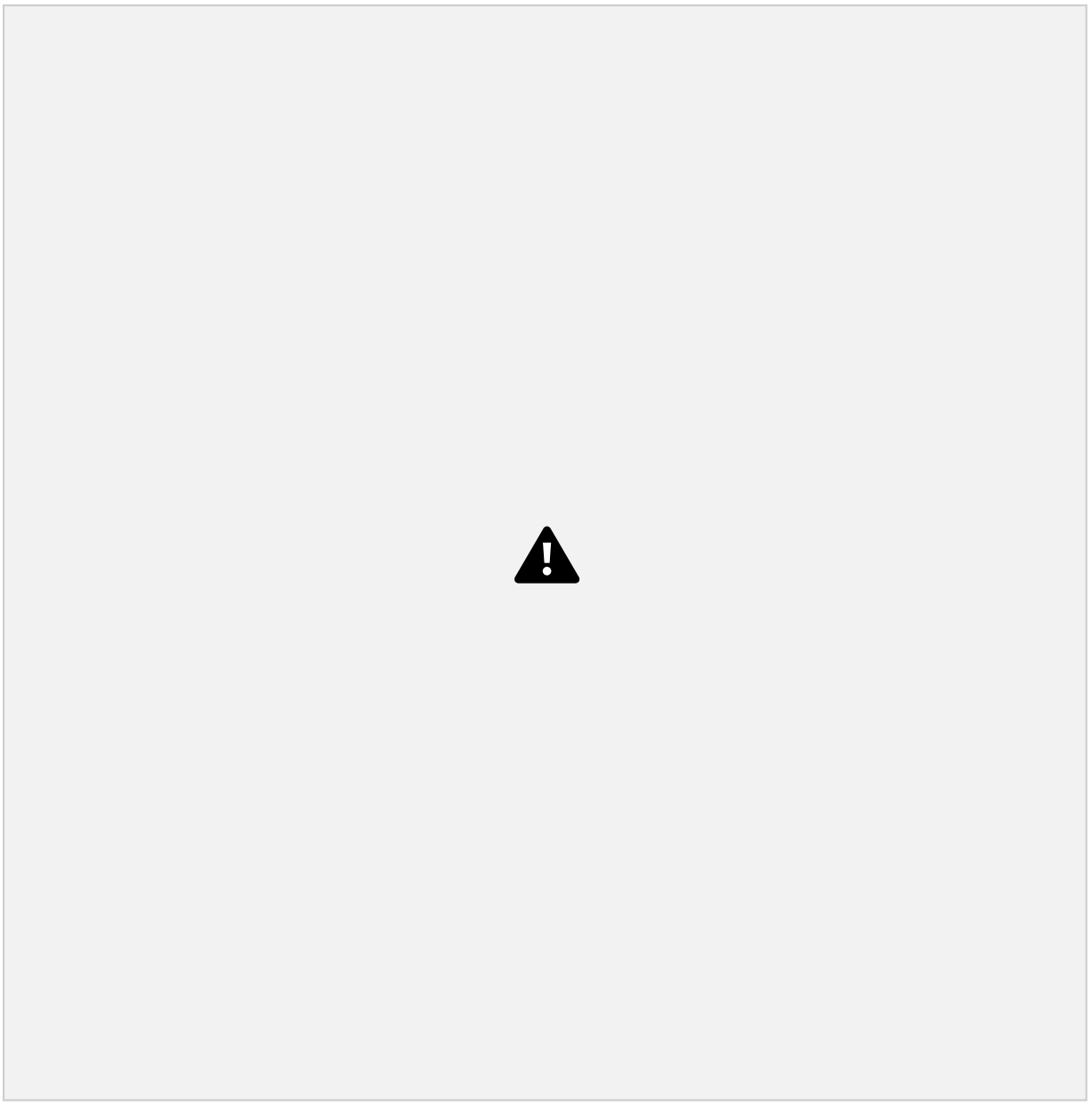
```
SELECT mc.IDProduto, p.Nome AS NomeProduto, SUM(mc.Quantidade
* mc.PrecoUnitario) AS ValorTotalEntrada
FROM MovimentoCompra mc
INNER JOIN Produtos p ON mc.IDProduto = p.IDProduto
GROUP BY mc.IDProduto, p.Nome;
```

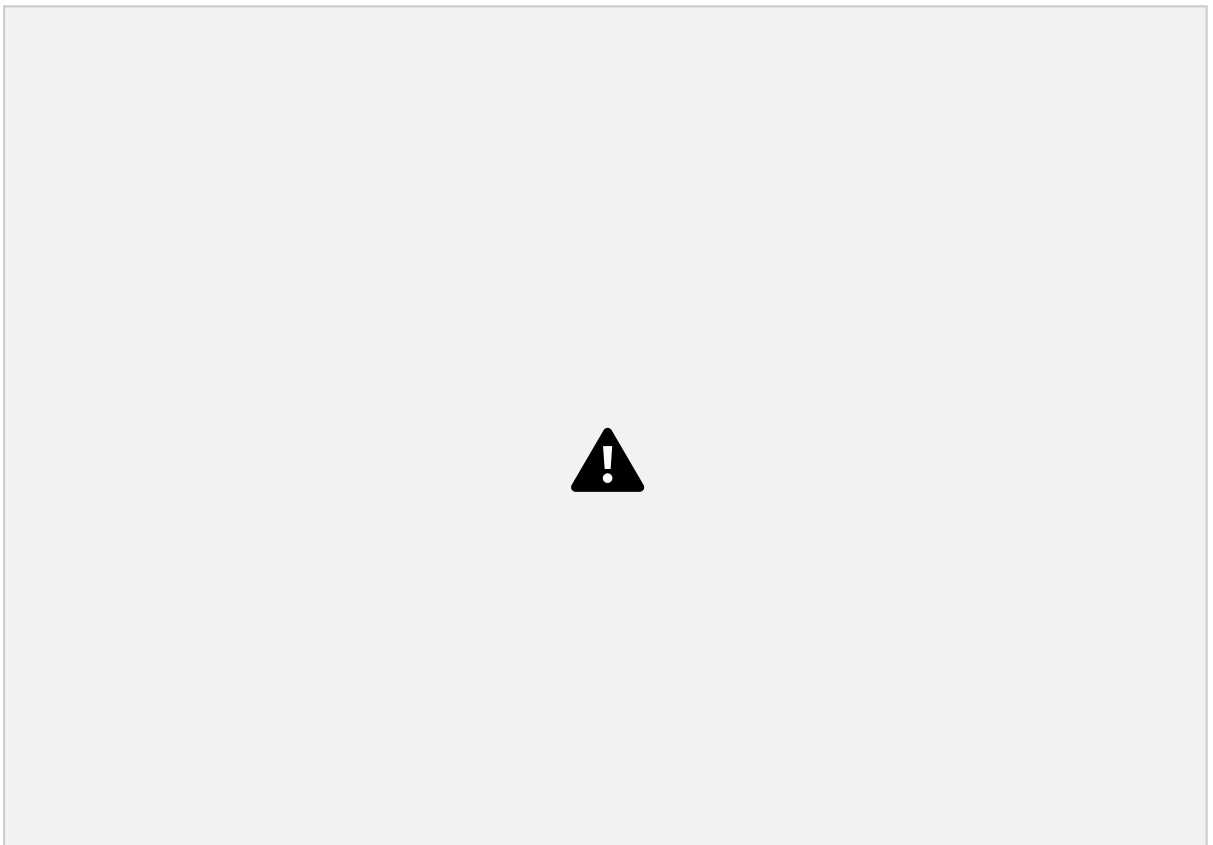




--Valor total das saídas agrupadas por produto:

```
SELECT mv.IDProduto, p.Nome AS NomeProduto, SUM(mv.Quantidade *
p.PrecoVenda) AS ValorTotalSaida
FROM MovimentoVenda mv
INNER JOIN Produtos p ON mv.IDProduto = p.IDProduto
GROUP BY mv.IDProduto, p.Nome;
```



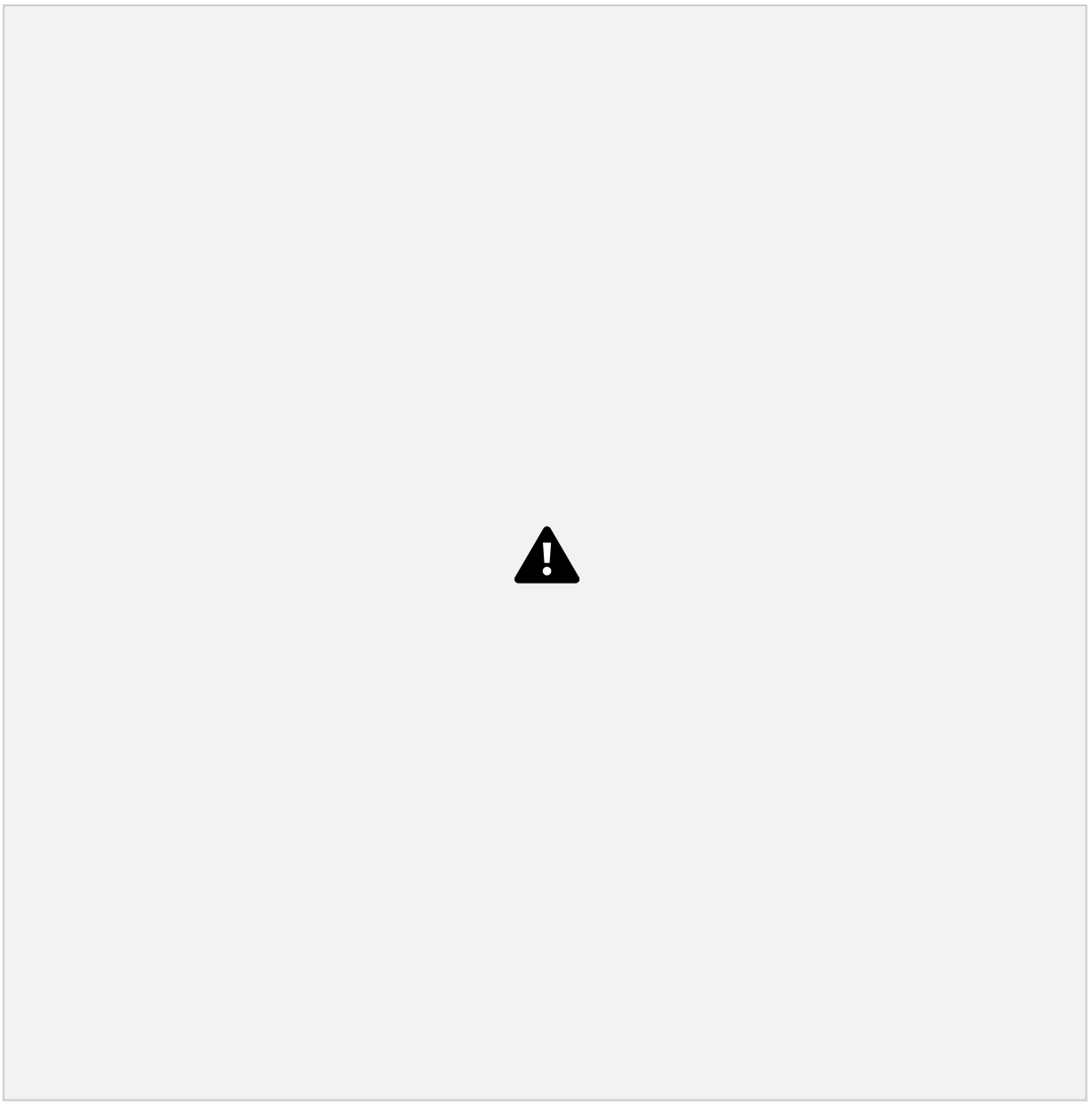


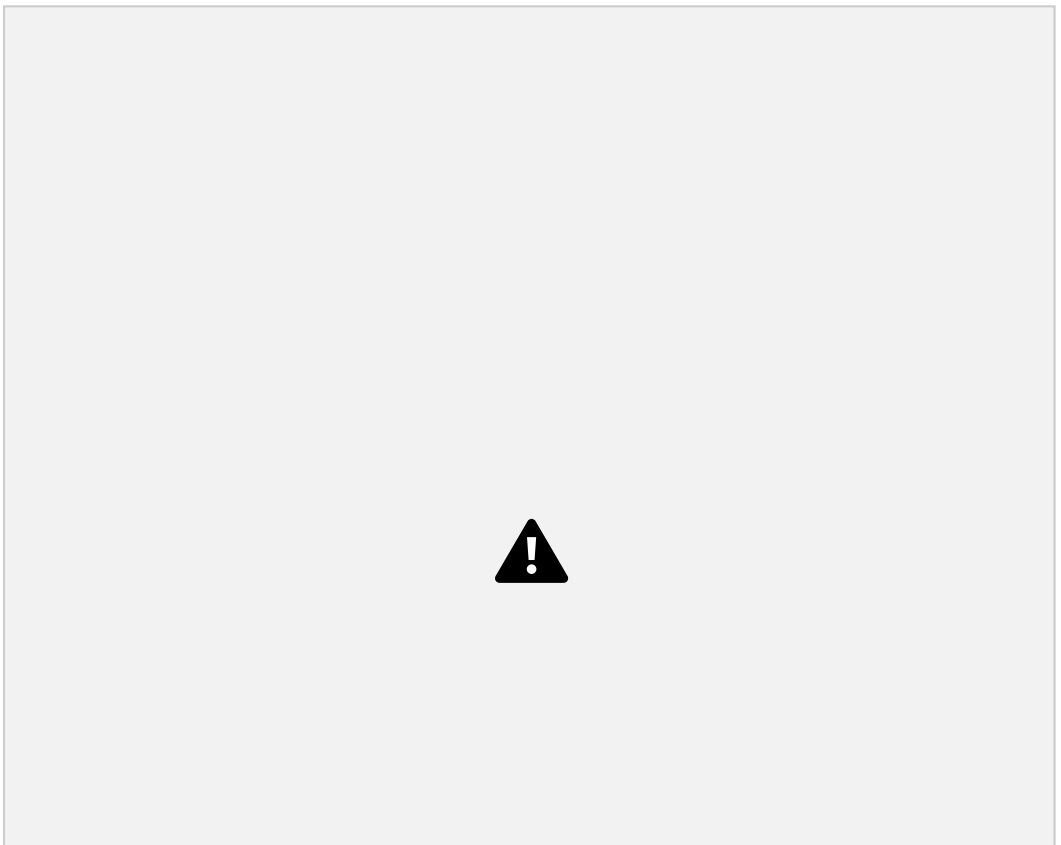
--Operadores que não efetuaram movimentações de entrada (compra):

```
SELECT u.IDUsuario, u.Nome AS NomeUsuario
FROM Usuarios u
LEFT JOIN MovimentoCompra mc ON u.IDUsuario = mc.Quantidade
WHERE mc.IDMovimento IS NULL;
```

dados pessoa fisica

```
SELECT mc.IDPessoaJuridica, pj.IDPessoa AS NomeFornecedor, SUM(mc.Quantidade
* mc.PrecoUnitario) AS ValorTotalEntrada
FROM MovimentoCompra mc
INNER JOIN PessoaJuridica pj ON mc.IDPessoaJuridica = pj.IDPessoa
GROUP BY mc.IDPessoaJuridica, pj.IDPessoa;
```





dados pessoa juridica

```
SELECT mv.IDPessoaFisica, u.Nome AS NomeUsuario, SUM(mv.Quantidade
* p.PrecoVenda) AS ValorTotalSaida
FROM MovimentoVenda mv
INNER JOIN Usuarios u ON mv.IDPessoaFisica = u.IDUsuario
INNER JOIN Produtos p ON mv.IDProduto = p.IDProduto
GROUP BY mv.IDPessoaFisica, u.Nome;
```

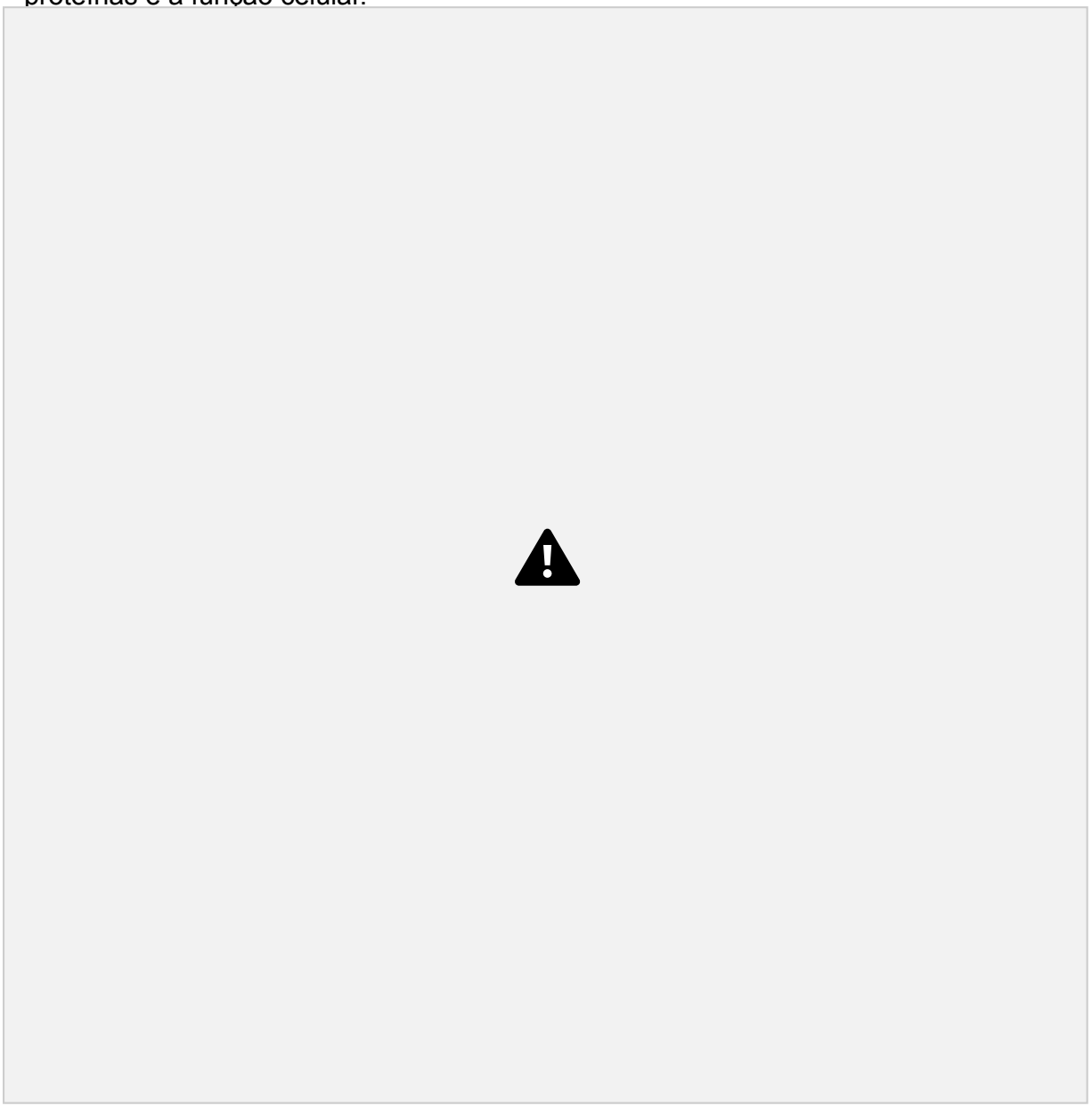
Análise e Conclusão:

Quais as diferenças no uso de sequence e identity?

"Sequence" e "identity" são dois termos que podem ser usados em diferentes contextos e têm significados diferentes, dependendo do contexto. Aqui, vou explicar as diferenças em seu uso comuns em dois contextos diferentes:

1. Biologia Molecular:

- **Sequence (Sequência)**: Nesse contexto, "sequence" refere-se à ordem específica de nucleotídeos em uma molécula de ácido nucleico, como o DNA ou o RNA. Uma sequência de DNA, por exemplo, é uma representação da ordem de adenina (A), citosina (C), guanina (G) e timina (T) em uma cadeia de DNA. Essa sequência é importante porque codifica as informações genéticas necessárias para a síntese de proteínas e a função celular.



- **Identity (Identidade)**: A "identity" em biologia molecular refere-se ao grau de

semelhança entre duas sequências de ácido nucleico ou proteína. Quando comparamos duas sequências, podemos calcular a identidade para determinar o quão parecidas são. Isso é frequentemente expresso como uma porcentagem, representando a proporção de nucleotídeos ou aminoácidos idênticos entre as duas sequências. Quanto maior a identidade, maior a semelhança entre as sequências.

2. Tecnologia da Informação (TI):

- **Sequence (Sequência)**: Em TI, "sequence" se refere a uma série de elementos (geralmente dados ou comandos) que estão dispostos em uma ordem específica. Uma sequência de caracteres em uma linguagem de programação, por exemplo, é uma série de caracteres organizados em uma ordem específica.

- **Identity (Identidade)**: Em TI, "identity" geralmente se refere à identificação única de um objeto, entidade ou usuário em um sistema. Isso pode ser feito por meio de identificadores únicos, como números de identificação, nomes de usuário ou chaves de autenticação. A "identity" é usada para garantir que o sistema possa distinguir entre diferentes elementos ou usuários.

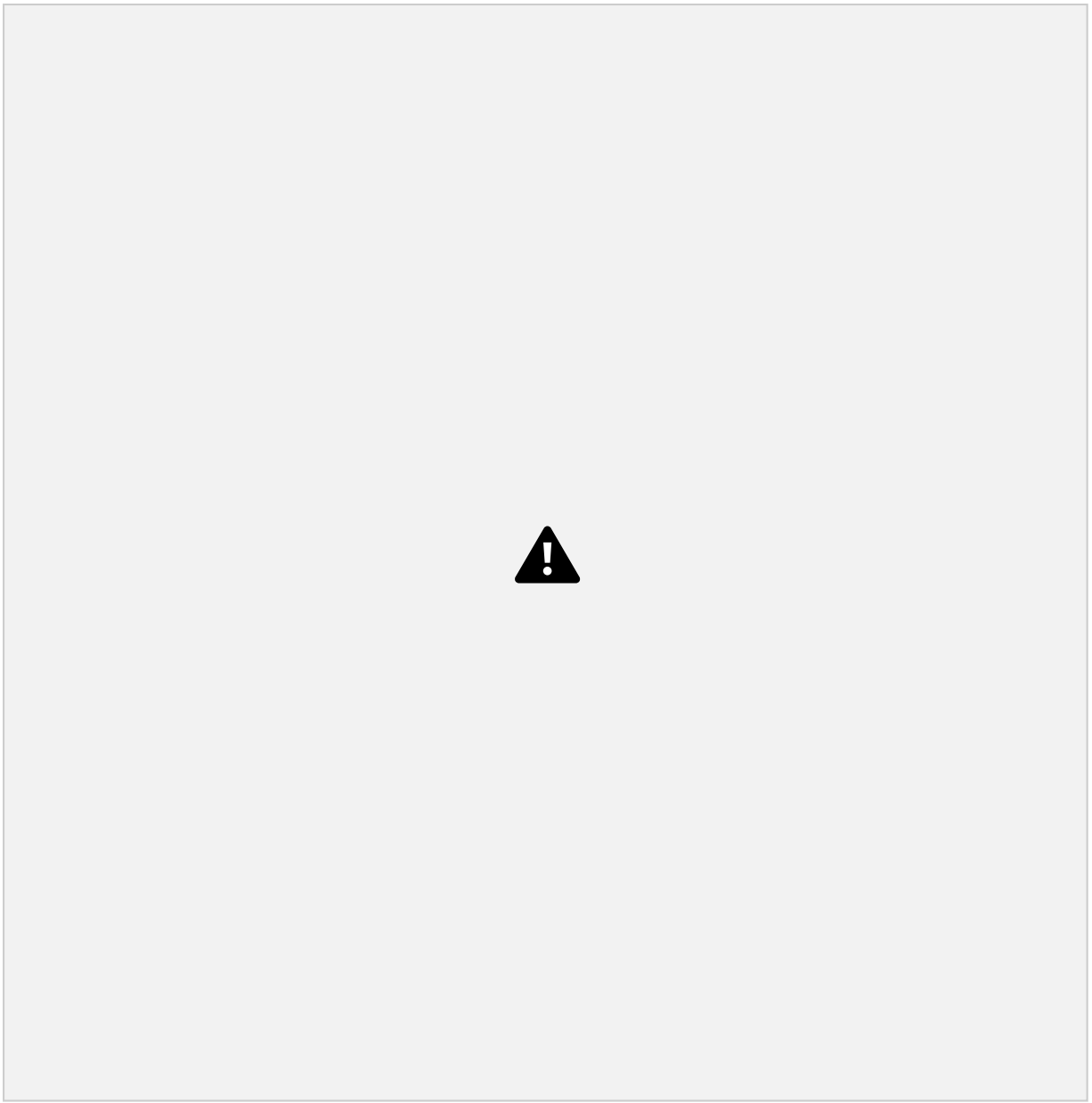
Portanto, as diferenças no uso de "sequence" e "identity" dependem do contexto em que esses termos são aplicados. Em biologia molecular, eles se referem à ordem de nucleotídeos e à semelhança entre sequências, enquanto em TI, eles se referem à organização de dados em uma ordem específica e à identificação única de elementos ou usuários.

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

As chaves estrangeiras (foreign keys) desempenham um papel fundamental na garantia da consistência dos dados em um banco de dados relacional. Elas são uma parte essencial da integridade referencial, que é uma característica-chave dos sistemas de gerenciamento de banco de dados (SGBDs). Aqui estão algumas das razões pelas quais as chaves estrangeiras são importantes para a consistência do banco de dados:

1. **Mantendo a Integridade Referencial**: Uma chave estrangeira define um relacionamento entre duas tabelas em um banco de dados. Ela estabelece uma conexão entre uma coluna em uma tabela (a chave estrangeira) e uma coluna em outra tabela (a chave primária), indicando que os valores na coluna da chave estrangeira devem corresponder aos valores na coluna da chave primária. Isso garante que os dados relacionados estejam corretos e consistentes.

2. **Evitando Dados Órfãos**: As chaves estrangeiras ajudam a evitar a existência de dados "órfãos" em uma tabela. Isso significa que, se uma linha em uma tabela referenciada (com a chave primária) for excluída, o SGBD pode ser configurado para



impedir que registros relacionados em outras tabelas sejam mantidos sem referências válidas. Isso evita registros inconsistentes e referências a dados que não existem mais.

3. **Garantindo Integridade de Dados**As chaves estrangeiras também ajudam a garantir a integridade dos dados, impedindo a inserção de valores inválidos em uma coluna de chave estrangeira. Se alguém tentar inserir um valor que não corresponda a uma chave primária na tabela referenciada, o SGBD pode impedir essa operação.

4. **Facilitando Consultas e Joins**Usar chaves estrangeiras torna mais fácil e eficiente realizar consultas que envolvem várias tabelas relacionadas. Isso é fundamental para a funcionalidade de um banco de dados relacional, pois permite a recuperação de informações de maneira coerente e organizada.

5. **Mantendo a Consistência dos Dados em Caso de Atualizações**Quando os dados são atualizados, as chaves estrangeiras ajudam a manter a consistência entre as

tabelas relacionadas. Se um valor na chave primária é atualizado, o SGBD pode ser configurado para atualizar automaticamente todas as referências a esse valor nas tabelas relacionadas.

6. **Documentação da Estrutura do Banco de Dados**: O uso de chaves estrangeiras também ajuda a documentar a estrutura do banco de dados, tornando mais claro como as tabelas estão relacionadas umas com as outras.

Em resumo, as chaves estrangeiras desempenham um papel crucial na manutenção da consistência e da integridade dos dados em um banco de dados relacional. Elas ajudam a garantir que os dados estejam organizados de maneira lógica e coerente, permitindo que o SGBD aplique regras e restrições que evitam erros e inconsistências nos dados.

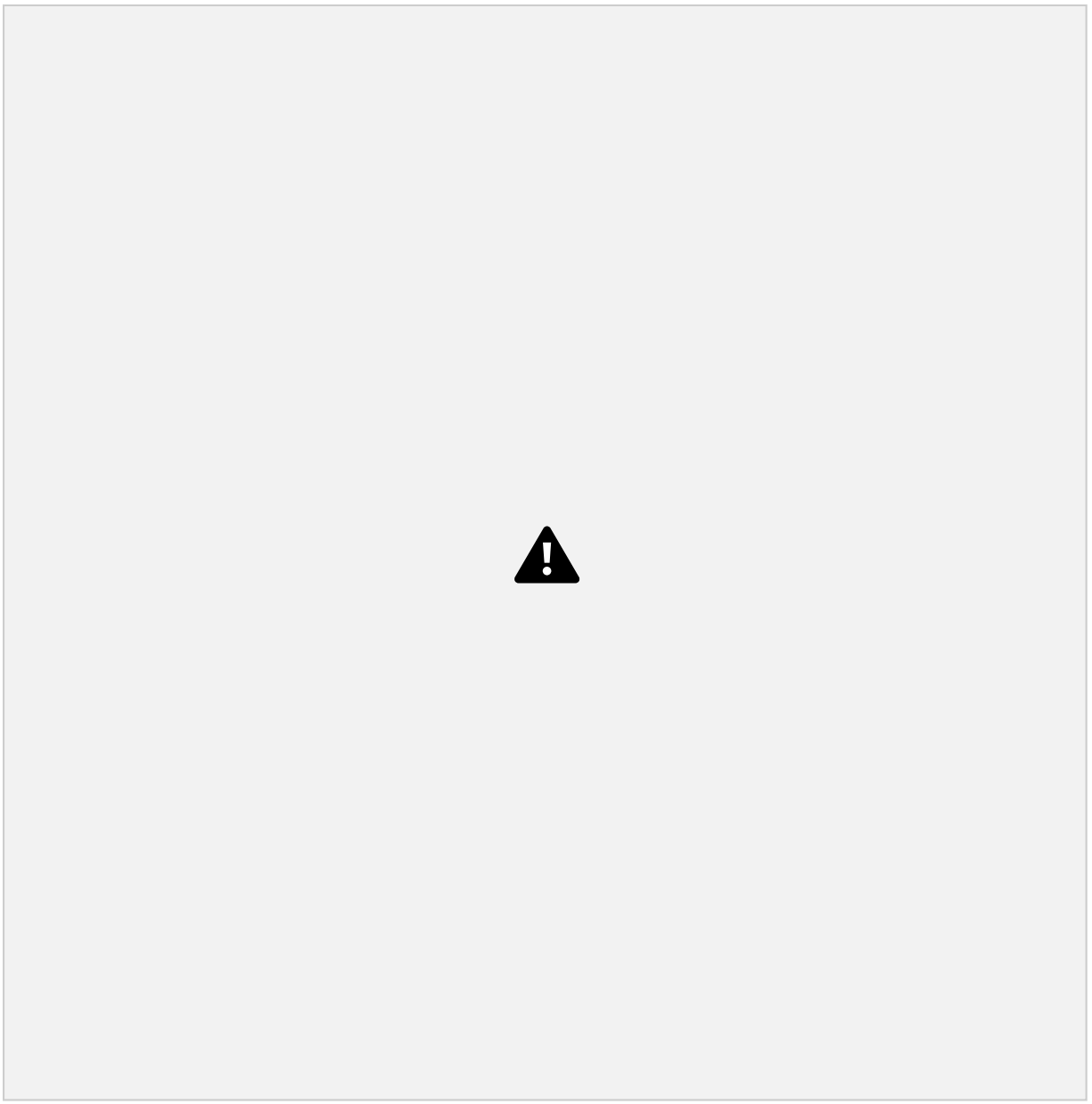
Quais operadores do SQL pertencem à álgebra relacional e quais são definidos no cálculo relacional?

O SQL (Structured Query Language) é uma linguagem de consulta usada em sistemas de gerenciamento de banco de dados relacionais (SGBDRs) e incorpora elementos tanto da álgebra relacional quanto do cálculo relacional. Vou listar alguns operadores e conceitos que pertencem a cada um desses modelos:

Operadores da Álgebra Relacional no SQL:
1. **Projeto (SELECT)**: O operador SELECT é usado para selecionar colunas específicas de uma tabela ou resultado de consulta.

2. **União (UNION)**: O operador UNION é usado para combinar as linhas de dois ou mais conjuntos de resultados em um único conjunto.

3. **Interseção (INTERSECT)**: O operador INTERSECT é usado para retornar apenas as linhas que estão presentes em ambos os conjuntos de resultados.



4. **Diferença (EXCEPT ou MINUS)**: O operador EXCEPT (ou MINUS em alguns bancos de dados) é usado para retornar as linhas que estão em um conjunto de resultados, mas não em outro.

5. **Produto Cartesiano (CROSS JOIN)**: O operador CROSS JOIN é usado para criar todas as combinações possíveis de linhas de duas tabelas, gerando um produto cartesiano.

6. **Restrição (WHERE)**: Embora não seja um operador da álgebra relacional clássica, a cláusula WHERE é usada no SQL para aplicar condições que filtram as linhas de uma tabela.

Operadores do Cálculo Relacional no SQL:
1. **Projeto (SELECT)**: Assim como na álgebra relacional, a cláusula SELECT do SQL também é usada para projetar colunas específicas de uma tabela.

2. **Restrição (WHERE)**: A cláusula WHERE no SQL pode ser vista como uma forma de seleção no cálculo relacional, pois permite filtrar as linhas com base em condições específicas.

É importante notar que, embora o SQL inclua elementos desses modelos, ele é uma linguagem mais abrangente e incorpora conceitos adicionais, como agrupamento (GROUP BY), agregação (SUM, COUNT, AVG, etc.), ordenação (ORDER BY), junção (JOIN), subconsultas (subqueries) e outras funcionalidades que não estão diretamente relacionadas à álgebra ou ao cálculo relacional tradicionais.

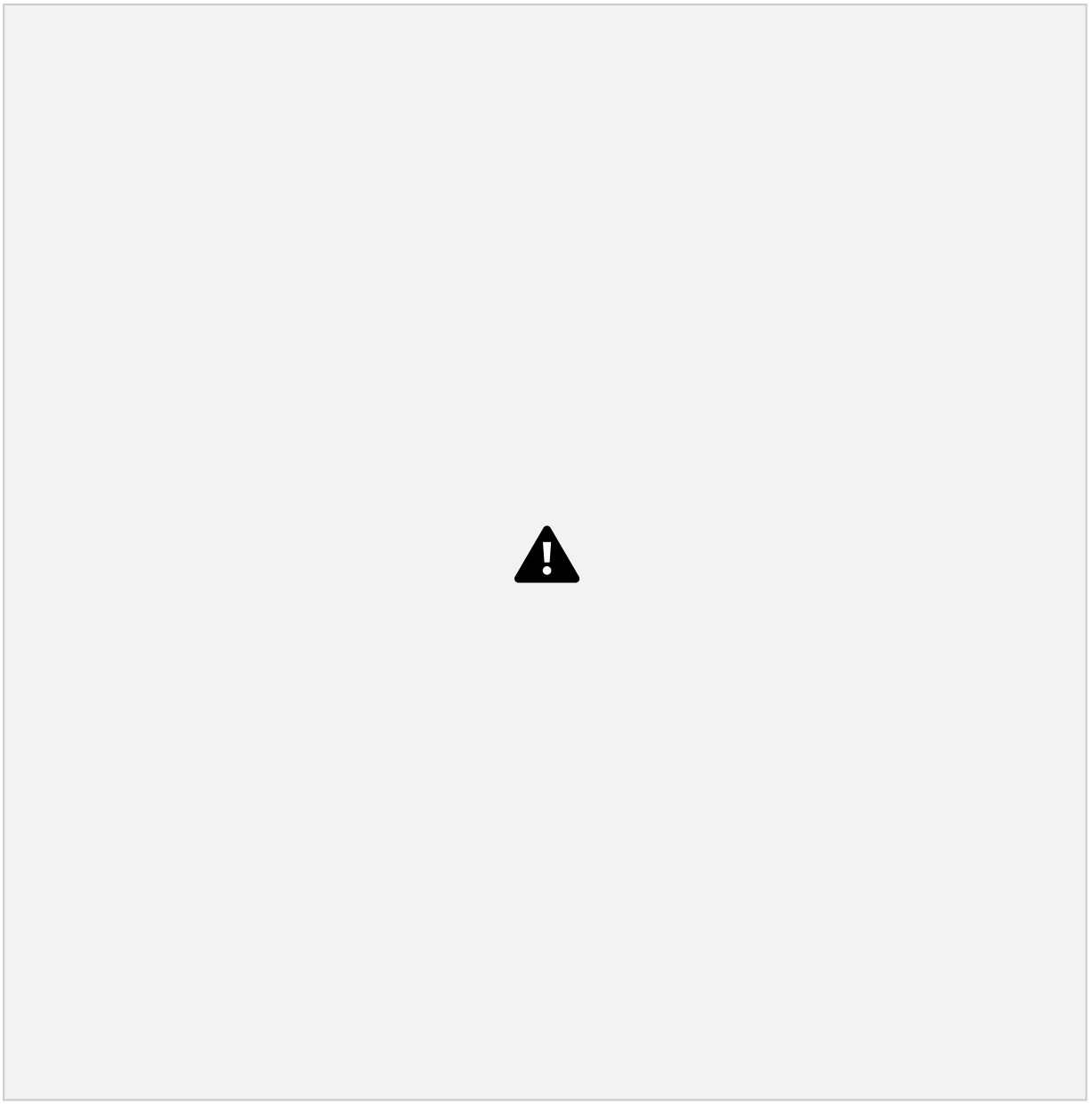
Em resumo, o SQL é uma linguagem de consulta que se baseia em conceitos da álgebra e do cálculo relacional, mas também inclui recursos adicionais para facilitar a recuperação e a manipulação de dados em bancos de dados relacionais.

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

O agrupamento em consultas SQL é realizado usando a cláusula `GROUP BY`. Esta cláusula é usada para agrupar linhas de uma tabela com base nos valores de uma ou mais colunas e, em seguida, aplicar funções de agregação a esses grupos de linhas. O resultado é um conjunto de resumo que representa os grupos e os resultados das funções de agregação para cada grupo.

Aqui está a sintaxe básica da cláusula `GROUP BY` em uma consulta SQL:

```sql
SELECT coluna1, coluna2, funcao_agregacao(coluna) AS alias
```



FROM tabela
GROUP BY coluna1, coluna2
```

Explicação dos elementos da consulta:

- `coluna1`, `coluna2`, etc.: São as colunas pelas quais você deseja agrupar os dados.

- `funcao_agregacao(coluna) AS alias`: Você pode aplicar funções de agregação como `SUM`, `COUNT`, `AVG`, `MAX`, `MIN`, etc., para calcular valores resumidos para cada grupo. O resultado é geralmente renomeado usando um alias.

- `tabela`: É a tabela da qual você está selecionando os dados.

Lembre-se de que, ao usar a cláusula `GROUP BY`, há um requisito importante que
```

deve ser atendido:

Todo coluna no SELECT que não está incluída em uma função de agregação deve estar listada na cláusula GROUP BY.

Isso significa que, se você está selecionando uma coluna que não é uma função de agregação, ela deve ser incluída na lista `GROUP BY`. Isso garante que o SGBD saiba como agrupar os dados corretamente. Por exemplo:

```sql
SELECT departamento, COUNT(*) AS total_funcionarios
FROM funcionarios
GROUP BY departamento
```

Neste exemplo, a coluna "departamento" é incluída tanto no SELECT quanto no GROUP BY, pois não é uma função de agregação. A função de agregação `COUNT(*)` calcula o número de funcionários em cada departamento.

O não cumprimento desse requisito pode resultar em erros de sintaxe ou em resultados imprecisos em sua consulta.

Portanto, ao usar a cláusula `GROUP BY`, lembre-se de agrupar corretamente os dados e aplicar funções de agregação às colunas que você deseja resumir. Isso é essencial para consultas de resumo e análise de dados em SQL.