Vamos manter informações

Ramon Barros Correa, 202208939635

Inserir aqui o Campus: Polo Sete lagoas (MG)
Nível 2 – Vamos manter informações, turma 2022.2, semestre 2024.1

Objetivo da Prática

O objetivo desta missão prática foi a criação de um banco de dados usando o Microsoft SQL Server, neste caso, foi usado inicialmente o SQL Server Express, mas posteriormente foi feito uma alteração para o SQL Server Developer, pois a versão anterior apresentava limitações para o seguimento de outras missões práticas

1º Procedimento | Criando o Banco de Dados

Inserir neste campo, **de forma organizada**, todos os códigos do roteiro do 1º Procedimento da Atividade Prática, os resultados da execução do código e a Análise e Conclusão:

- Criando tabela produtos

```
CREATE TABLE Produtos (
   ID INTEGER     ,
   Nome VARCHAR(MAX)     ,
   Quantidade INTEGER     ,
   Preco DECIMAL     ,
PRIMARY KEY(ID));
```

- Criando tabela Pessoas

```
CREATE TABLE Pessoas (
   ID INTEGER     ,
   Nome VARCHAR(MAX)     ,
   Endereco VARCHAR(MAX)     ,
   Telefone VARCHAR(MAX)     ,
   Usuarios_ID INTEGER     ,
   MovimentosVenda_ID INTEGER    ,
PRIMARY KEY(ID)     ,
  FOREIGN KEY(MovimentosVenda_ID)
    REFERENCES MovimentosVenda(ID),
  FOREIGN KEY(Usuarios_ID)
    REFERENCES Usuarios(ID));


CREATE INDEX Pessoas_FKIndex1 ON Pessoas (MovimentosVenda_ID);
CREATE INDEX Pessoas_FKIndex2 ON Pessoas (Usuarios_ID);


CREATE INDEX IFK_Rel_04 ON Pessoas (MovimentosVenda_ID);
CREATE INDEX IFK_Rel_10 ON Pessoas (Usuarios_ID);
```

- Criando tabela de pessoas juridicas

```
CREATE TABLE PessoasJuridicas (
  ID INTEGER    ,
  Pessoas_ID INTEGER    ,
  CNPJ VARCHAR(MAX)        ,
PRIMARY KEY(ID)   ,
  FOREIGN KEY(Pessoas_ID)
    REFERENCES Pessoas(ID));


CREATE INDEX PessoasJuridicas_FKIndex1 ON PessoasJuridicas (Pessoas_ID);


CREATE INDEX IFK_Rel_02 ON PessoasJuridicas (Pessoas_ID);
```

- Criando tabela de Pessoas fisicas

```
CREATE TABLE PessoasFisicas (
    ID INTEGER     ,
  Pessoas_ID INTEGER    ,
  CPF VARCHAR(MAX)       ,
PRIMARY KEY( ID)   ,
  FOREIGN KEY(Pessoas_ID)
    REFERENCES Pessoas(ID));


CREATE INDEX PessoasFisicas_FKIndex1 ON PessoasFisicas (Pessoas_ID);


CREATE INDEX IFK_Rel_03 ON PessoasFisicas (Pessoas_ID);
```

- Criação da tabela de Movimentos

```
CREATE TABLE MovimentosCompra (
  ID INTEGER    ,
  Usuarios_ID INTEGER    ,
  Produtos_ID INTEGER    ,
  ID_Usuario INTEGER    ,
  ID_Produto INTEGER    ,
  ID_PessoaJuridica INTEGER     ,
  Quantidade INTEGER     ,
  PrecoUnitario DECIMAL       ,
PRIMARY KEY(ID)    ,
  FOREIGN KEY(Produtos_ID)
    REFERENCES Produtos(ID),
  FOREIGN KEY(Usuarios_ID)
    REFERENCES Usuarios(ID));


CREATE INDEX MovimentosCompra_FKIndex1 ON MovimentosCompra (Produtos_ID);
CREATE INDEX MovimentosCompra_FKIndex2 ON MovimentosCompra (Usuarios_ID);


CREATE INDEX IFK_Rel_06 ON MovimentosCompra (Produtos_ID);
CREATE INDEX IFK_Rel_09 ON MovimentosCompra (Usuarios_ID);
```

```
CREATE TABLE MovimentosVenda (
  ID INTEGER    ,
  Usuarios_ID INTEGER    ,
  Produtos_ID INTEGER    ,
  ID_Usuario INTEGER    ,
  ID_Produto INTEGER    ,
  ID_PessoaFisica INTEGER     ,
  Quantidade INTEGER     ,
  PrecoUnitario DECIMAL       ,
PRIMARY KEY(ID)    ,
  FOREIGN KEY(Produtos_ID)
    REFERENCES Produtos(ID),
  FOREIGN KEY(Usuarios_ID)
    REFERENCES Usuarios(ID));


CREATE INDEX MovimentosVenda_FKIndex1 ON MovimentosVenda (Produtos_ID);
CREATE INDEX MovimentosVenda_FKIndex2 ON MovimentosVenda (Usuarios_ID);


CREATE INDEX IFK_Rel_07 ON MovimentosVenda (Produtos_ID);
CREATE INDEX IFK_Rel_08 ON MovimentosVenda (Usuarios_ID);
```

- Criado tabela de usuarios

```
CREATE TABLE Usuarios (
  ID INTEGER    ,
  Nome VARCHAR(MAX)    ,
  Senha VARCHAR(MAX)     ,
  Papel VARCHAR(MAX)       ,
PRIMARY KEY(ID));
```

- Criando sequencia

```
CREATE SEQUENCE PessoaSequence
    START WITH 0
    INCREMENT BY 1;
```

a)Como são implementadas as diferentes cardinalidades, basicamente 1x1, 1xN, ou NxN em um banco de dados relacional?

1 para 1 (1x1):
Nessa relação, uma entidade em uma tabela está associada a exatamente uma entidade em outra tabela e vice-versa.
1 para muitos (1xN):
Nessa relação, uma entidade em uma tabela pode estar associada a uma ou mais entidades em outra tabela, mas cada entidade na segunda tabela está associada a no máximo uma entidade na primeira tabela.
Muitos para muitos (NxN):
Nessa relação, cada entidade em uma tabela pode estar associada a várias entidades em outra tabela e vice-versa.

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

Tabela por subclasse (ou tabela por tipo):
Nessa abordagem, cada subclasse tem sua própria tabela no banco de dados. A tabela da superclasse contém os atributos comuns a todas as subclasses, enquanto as tabelas das subclasses contêm apenas os atributos específicos de cada uma.
Cada tabela de subclasse possui uma chave estrangeira que faz referência à tabela da superclasse, estabelecendo assim a relação de herança.

Tabela única (ou tabela única com campos nulos):
Nessa abordagem, todos os atributos de todas as subclasses são armazenados em uma única tabela.
São adicionados campos para cada atributo de cada subclasse, e os campos que não se aplicam a uma determinada subclasse são preenchidos com valores nulos.

Tabela por hierarquia (ou tabela por superclasse):
Nessa abordagem, todas as entidades (tanto a superclasse quanto as subclasses) são armazenadas em uma única tabela.
São adicionados campos para identificar o tipo de cada entidade (superclasse ou subclasse), e os atributos específicos de cada subclasse são armazenados em colunas separadas.

c)Como o SQL Server Management Studio permite a melhoria para produtividade nas tarefas relacionadas?

O SQL Server Management Studio (SSMS) oferece várias funcionalidades e recursos que ajudam a melhorar a produtividade nas tarefas relacionadas ao gerenciamento e desenvolvimento de bancos de dados no SQL Server, Como: Interface Gráfica Intuitiva, Editor de Consultas Avançado, Assistente de Importação e Exportação de Dados, Designer de Tabelas e Diagramas, etc.

2º Procedimento | Alimentando a Base

Inserir neste campo, **de forma organizada**, todos os códigos do roteiro do 2º Procedimento da Atividade Prática, os resultados da execução do código e a Análise e Conclusão:

- Inserindo dados na tabela Pessoas\PessoasFisicas\PessoasJuridicas

```
INSERT INTO Pessoas (ID, Nome, Endereco, Telefone)
VALUES (NEXT VALUE FOR Seq_Pessoa_ID, 'joao', 'rua dos palmeiras', '11992999');

INSERT INTO Pessoas (ID, Nome, Endereco, Telefone)
VALUES (NEXT VALUE FOR Seq_Pessoa_ID, 'JC', 'rua dos altos', '3137121123');

INSERT INTO PessoasFisicas (ID, CPF)
VALUES ('1', '123.456.789-00');

INSERT INTO PessoasJuridicas (ID, CNPJ)
VALUES ('2', '12.345.678/0001-00');
```

- Inserindo dados na tabela de usuários:

```
INSERT INTO Usuarios (ID, Nome, Senha, Papel)
VALUES
('1', 'op1', 'op1', 'Operador'),
('2', 'op2', 'op2', 'Operador');
```

- Inserindo dados na tabela de produtos

```
INSERT INTO Produtos (ID, Nome, Quantidade, Preco)
VALUES
(1, 'Produto A', 100, 10.99),
(2, 'Produto B', 50, 20.49),
(3, 'Produto C', 200, 5.99);
```

- Inserindo dados na tabela movimentos

```
-- Inserir movimentações de compra
INSERT INTO MovimentosCompra (ID, ID_Usuario, ID_Produto, ID_PessoaJuridica, Quantidade, PrecoUnitario)
VALUES
(1, 1, 1, 1, 10, 15.99),  -- Exemplo de compra do produto ID 1 (10 unidades por R$ 15.99)
(2, 2, 2, 2, 5, 25.49);     -- Exemplo de compra do produto ID 2 (5 unidades por R$ 25.49)


-- Inserir movimentações de venda
INSERT INTO MovimentosVenda (ID, ID_Usuario, ID_Produto, ID_PessoaFisica, Quantidade, PrecoUnitario)
VALUES
(1, 1, 1, 1, 3, 19.99),   -- Exemplo de venda do produto ID 1 (3 unidades por R$ 19.99)
(2, 2, 2, 2, 2, 29.99);    -- Exemplo de venda do produto ID 2 (2 unidades por R$ 29.99)
```

- Exibindo dados da tabela pessoas físicas e juridicas

```
SELECT *
FROM PessoasFisicas;

SELECT *
FROM PessoasJuridicas;
```

- Exibindo entradas e saida

```sql
SELECT MC.ID AS MovimentoID, P.Nome AS Produto, PJ.Nome AS Fornecedor, MC.Quantidade, MC.PrecoUnitario, MC.Quantidade * MC.PrecoUnitario AS ValorTotal
FROM MovimentosCompra MC
JOIN Produtos P ON MC.ID_Produto = P.ID
JOIN PessoasJuridicas PJ ON MC.ID_PessoaJuridica = PJ.ID;

SELECT MV.ID AS MovimentoID, P.Nome AS Produto, PF.Nome AS Comprador, MV.Quantidade, MV.PrecoUnitario, MV.Quantidade * MV.PrecoUnitario AS ValorTotal
FROM MovimentosVenda MV
JOIN Produtos P ON MV.ID_Produto = P.ID
JOIN PessoasFisicas PF ON MV.ID_PessoaFisica = PF.ID;
```

Valor total das entradas por produto

```sql
SELECT P.Nome AS Produto, SUM(MC.Quantidade * MC.PrecoUnitario) AS TotalEntrada
FROM MovimentosCompra MC
JOIN Produtos P ON MC.ID_Produto = P.ID
GROUP BY P.Nome;
```

Valor total das saidas por produto

```sql
SELECT P.Nome AS Produto, SUM(MV.Quantidade * MV.PrecoUnitario) AS TotalSaida
FROM MovimentosVenda MV
JOIN Produtos P ON MV.ID_Produto = P.ID
GROUP BY P.Nome;
```

Operadores que não efetuaram movimentações de entrada

```sql
SELECT *
FROM Usuarios U
WHERE NOT EXISTS (
    SELECT 1
    FROM MovimentosCompra MC
    WHERE MC.ID_Usuario = U.ID
);
```

Valor total de entrada, agrupado por operador.

```sql
SELECT U.Nome AS Operador, SUM(MC.Quantidade * MC.PrecoUnitario) AS TotalEntrada
FROM MovimentosCompra MC
JOIN Usuarios U ON MC.ID_Usuario = U.ID
GROUP BY U.Nome;
```

Valor total de saída, agrupado por operador.

```sql
SELECT U.Nome AS Operador, SUM(MV.Quantidade * MV.PrecoUnitario) AS TotalSaida
FROM MovimentosVenda MV
JOIN Usuarios U ON MV.ID_Usuario = U.ID
GROUP BY U.Nome;
```

Valor médio de venda por produto, utilizando média ponderada.

```sql
SELECT P.Nome AS Produto, SUM(MV.Quantidade * MV.PrecoUnitario) / SUM(MV.Quantidade) AS ValorMedioVenda
FROM MovimentosVenda MV
JOIN Produtos P ON MV.ID_Produto = P.ID
GROUP BY P.Nome;
```

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

Sequências (sequences):
As sequências são objetos de banco de dados independentes que geram uma série de valores únicos em ordem crescente ou decrescente. Elas não estão diretamente associadas a uma tabela específica; em vez disso, podem ser usadas por várias tabelas em um banco de dados. As sequências são definidas separadamente do esquema da tabela e podem ser usadas em qualquer tabela que necessite de valores únicos. No SQL Server, você cria uma sequência usando a instrução CREATE SEQUENCE e pode recuperar o próximo valor da sequência usando a função NEXT VALUE FOR.

Colunas de Identidade (identity columns):
As colunas de identidade são colunas de uma tabela que são automaticamente preenchidas com valores únicos à medida que novas linhas são inseridas na tabela. Elas são específicas de uma tabela e geralmente são usadas para criar chaves primárias ou outras colunas que precisam de valores únicos automaticamente gerados. No SQL Server, você define uma coluna de identidade adicionando a propriedade IDENTITY à definição da coluna ao criar a tabela. O valor de uma coluna de identidade é gerenciado automaticamente pelo SQL Server e não pode ser controlado manualmente. O SQL Server atribui automaticamente valores crescentes à medida que novas linhas são inseridas na tabela.

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

Mantém a integridade referencial, Impede a exclusão ou modificação de dados relacionados de forma não intencional, Facilita a manutenção e a compreensão do esquema do banco de dados, Melhora a eficiência das consultas e Promove a consistência dos dados.

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

Na álgebra relacional, os operadores SQL são baseados em operações de conjunto e operações de projeção, seleção, união, interseção e diferença de relações. Alguns dos operadores SQL mais comuns que pertencem à álgebra relacional incluem:

Seleção (Selection): Filtra as tuplas de uma relação que atendem a uma condição específica. Em SQL, isso é realizado com a cláusula WHERE.
Projeção (Projection): Seleciona colunas específicas de uma relação. Em SQL, isso é realizado com a cláusula SELECT.
União (Union): Combina duas relações em uma única relação que contém todas as tuplas das relações originais, eliminando duplicatas. Em SQL, isso é realizado com o operador UNION.
Interseção (Intersection): Retorna uma relação que contém apenas as tuplas que são comuns a duas relações. Em SQL, isso pode ser simulado usando junções e cláusulas WHERE.
Diferença (Difference): Retorna uma relação que contém as tuplas presentes em uma relação, mas não em outra. Em SQL, isso é realizado com o operador EXCEPT ou usando junções e cláusulas WHERE.

Por outro lado, o cálculo relacional define operadores de forma mais descritiva, em vez de manipular diretamente as relações como na álgebra relacional. Alguns dos operadores definidos no cálculo relacional incluem:

Seleção (Selection): Seleciona tuplas que atendem a uma condição específica, semelhante à álgebra relacional. Projeção (Projection): Seleciona atributos específicos de uma relação, semelhante à álgebra relacional. Junção (Join): Combina duas ou mais relações com base em uma condição de junção.
Renomeação (Renaming): Renomeia os atributos de uma relação.

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

O agrupamento em consultas SQL é feito usando a cláusula GROUP BY, que permite agrupar as linhas de um conjunto de resultados com base em valores comuns em uma ou mais colunas. O requisito obrigatório para usar a cláusula GROUP BY é que todas as colunas selecionadas na lista de projeção devem ser agregadas ou incluídas na cláusula GROUP BY.