



Estácio

Universidade Estácio

Campus Vargem Grande Paulista

Curso: Desenvolvimento Full Stack

Disciplina: Nível 2 vamos manter as informações?

Turma: 2024.3

3º semestre letivo

Marcia da Silva e Souza

Título da prática: Vamos manter as informações!

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

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.

Relatório discente de acompanhamento -1º Procedimento | Criando o Banco de Dados

Códigos solicitados:

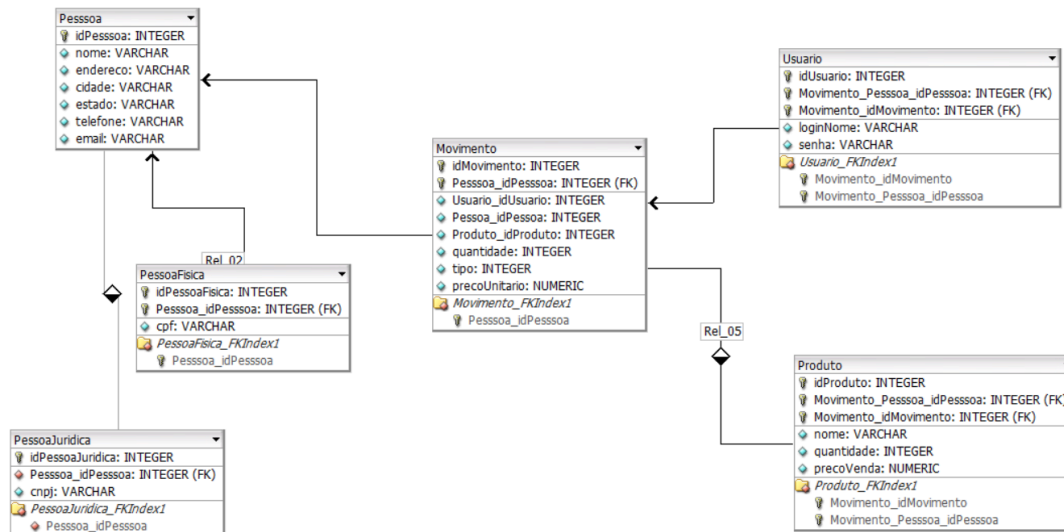


DIAGRAMA DE RELACIONAMENTO (DER)

SQLQuery1.sql:

```
USE Loja;  
GO
```

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

```
CREATE TABLE Pessoa(  
    idPessoa: INTEGER  
    nome: VARCHAR  
    endereco: VARCHAR  
    cidade: VARCHAR  
    estado: VARCHAR  
    telefone: VARCHAR  
    email: VARCHAR
```

```
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
```

```
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
```

```
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
```

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

```
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
```

SQLQuery3.sql:

```
INSERT INTO
Pessoa(idPessoa,nome,endereco,cidade,estado,telefone,email
)
```

```
VALUES (NEXT VALUE FOR orderPessoa, 'Andrea','Avenida
A, 11','Rio Branco','AC','1111-1111','andre@gmail.com'),
(NEXT VALUE FOR orderPessoa, 'Bruna','Avenida B,
22','Salvador','BA','2222-2222','bruna@gmail.com'),
(NEXT VALUE FOR orderPessoa, 'Carlos','Avenida C,
33','Fortaleza','CE','3333-3333','carlos@gmail.com'),
(NEXT VALUE FOR orderPessoa, 'Distribuidora
Delta','Avenida D,
44','Brasilia','DF','4444-4444','delta@gmail.com'),
(NEXT VALUE FOR orderPessoa, 'Empresa Echo','Avenida
E, 55','Vitoria','ES','5555-5555','echo@gmail.com');
```

```
INSERT INTO PessoaFisica(FK_Pessoa_idPessoa,cpf)
VALUES (1,'11111111111'),
(2,'22222222222'),
(3,'33333333333');
```

```
INSERT INTO PessoaJuridica(FK_Pessoa_idPessoa,cnpj)
VALUES (4,'4444444444444444'),
(5,'5555555555555555');
```

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

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

```
INSERT INTO
Movimento(FK_Usuario_idUsuario,FK_Pessoa_idPessoa,FK_P
roduto_idProduto,quantidade,tipo,precoUnitario)
```

```
VALUES (1,1,1,10,'E',5.00),  
       (2,2,2,20,'S',2.00),  
       (1,3,3,30,'E',4.00);
```

SQLQuery4.sql:

-- item (a)

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

-- item (b)

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

-- item (c)

```
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';
```

-- item (d)

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

-- item (e)

```
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;
```

-- item (f)

```
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;
```

-- item (g)

```
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;
```

```
-- item (h)  
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;
```

```
-- item (i)  
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;
```

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

]

Análise e conclusão:

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

R:Para a definição da cardinalidade de um-para-um (1X1), é necessário criar 2 tabelas, estabelecer uma chave primária na tabela 1 e uma chave estrangeira na tabela 2, com referência para chave primária da tabela 1.

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

R:Em bancos de dados relacionais, o tipo de relacionamento utilizado para representar herança é a generalização/especialização. Essa técnica permite organizar as entidades em uma hierarquia, onde uma entidade superclasse (geralmente mais abrangente) pode ter uma ou mais entidades subclasse (mais específicas). Existem três tipos principais de generalização/especialização:

- Generalização total: ○ A superclasse pode existir independentemente das subclasses. ○ Todas as entidades da superclasse devem pertencer a pelo menos uma subclasse. ○ Exemplo: Pessoa (superclasse) → Cliente e Funcionário (subclasses).
- Generalização parcial: ○ A superclasse não pode existir independentemente das subclasses. ○ Algumas entidades da superclasse podem não pertencer a nenhuma subclasse. ○ Exemplo: Animal (superclasse) → Cachorro e Gato (subclasses).
- Herança múltipla: ○ Uma subclasse pode herdar mais de uma superclasse. ○ Pode ser mais complexa de implementar e gerenciar. ○ Exemplo: Aluno (subclasse) herda de Pessoa e Curso (superclasses).

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

R:Na álgebra relacional, alguns operadores comuns incluem:

1. Seleção (σ): Filtra linhas com base em um critério.
2. Projeção (π): Seleciona colunas específicas.
3. União (\cup): Combina duas tabelas.
4. Interseção: Encontra linhas comuns entre duas tabelas.
5. Diferença: Encontra linhas em uma tabela que não estão em outra.
6. Produto Cartesiano (\times): Combina todas as linhas de duas tabelas.
7. Junção (\bowtie): Combina tabelas com base em uma condição.

No cálculo relacional, os operadores são mais relacionados a fórmulas e expressões lógicas:

1 Existencial (\exists): Verifica se há pelo menos um elemento que satisfaz uma condição.

1. Universal (\forall): Verifica se todos os elementos satisfazem uma condição.
2. Negação (\neg): Invertem a condição.
3. Conjunção (\wedge): Combina condições com "e".
4. Disjunção (\vee): Combina condições com "ou".

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

R:O agrupamento em consultas SQL é feito usando a cláusula GROUP BY. Ele agrupa linhas que têm valores em comum em colunas específicas, permitindo a realização de funções de agregação, como SUM(), AVG(), COUNT(), MIN(), e MAX().

O requisito obrigatório é que todas as colunas não agregadas na SELECT devem ser incluídas na cláusula GROUP BY. Aqui está um exemplo básico:

Relatório discente de acompanhamento - 2º

Procedimento | Alimentando a base

2. 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.

Análise e conclusão:

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

R: Sequences e identities são ambos usados para gerar valores únicos em colunas de banco de dados, mas eles têm algumas diferenças importantes.

IDENTITY:

Atrelado a uma Tabela: O IDENTITY é especificado durante a criação de uma tabela e está diretamente ligado a uma coluna dessa tabela.

Incremento Automático: Sempre que uma nova linha é inserida na tabela, o valor da coluna IDENTITY é incrementado automaticamente.

Limitação de Uso: O IDENTITY só pode ser usado para uma tabela por vez e é específico para essa coluna.

SEQUENCE:

Objeto Independente: Um SEQUENCE é um objeto separado no banco de dados, não atrelado a nenhuma tabela específica.

Flexibilidade: Pode ser usado para gerar valores únicos que podem ser compartilhados entre várias tabelas e colunas.

Customização: Você pode configurar o intervalo, o valor inicial e o incremento do SEQUENCE.

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

As chaves estrangeiras são cruciais para a consistência do banco de dados, pois garantem que os relacionamentos entre

as tabelas permaneçam válidas. Elas ajudam a manter a integridade referencial, ou seja:

Evita dados órfãos: Impede que existam registros em uma tabela que não tenham uma correspondência válida na tabela relacionada.

Assegura atualizações consistentes: Garante que as mudanças nas tabelas relacionadas sejam refletidas em todas as tabelas relevantes.

Facilita a manutenção de integridade: Ajuda a manter a consistência e a qualidade dos dados, evitando inconsistências no banco.

Fortalece as regras de negócio: Assegura que as relações definidas entre tabelas sigam as regras de negócios.

Em resumo, chaves estrangeiras são como guardiões que asseguram que todos os registros em suas tabelas estão corretamente relacionados e consistentes com outras tabelas.

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

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

Copilot

Na álgebra relacional, alguns operadores comuns incluem:

Seleção (σ): Filtra linhas com base em um critério.

Projeção (π): Seleciona colunas específicas.

União (\cup): Combina duas tabelas.

Interseção: Encontra linhas comuns entre duas tabelas.

Diferença: Encontra linhas em uma tabela que não estão em outra.

Produto Cartesiano (\times): Combina todas as linhas de duas tabelas.

Junção (\bowtie): Combina tabelas com base em uma condição.

No cálculo relacional, os operadores são mais relacionados a fórmulas e expressões lógicas:

Existencial (\exists): Verifica se há pelo menos um elemento que satisfaz uma condição.

Universal (\forall): Verifica se todos os elementos satisfazem uma condição.

Negação (\neg): Invertem a condição.

Conjunção (\wedge): Combina condições com "e".

Disjunção (\vee): Combina condições com "ou".

Esses operadores são usados para realizar consultas e manipulações de dados nos bancos de dados relacionais.

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

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

Copilot

O agrupamento em consultas SQL é feito usando a cláusula GROUP BY. Ele agrupa linhas que têm valores em comum em colunas específicas, permitindo a realização de funções de agregação, como SUM(), AVG(), COUNT(), MIN(), e MAX().

O requisito obrigatório é que todas as colunas não agregadas na SELECT devem ser incluídas na cláusula GROUP BY.