Curso - Desenvolvimento Full Stack Disciplina – Iniciando o caminho pelo Java

Lucas Pietro Santos de Souza Matricula – 2022 0809 0648

Campus - Polo Duque de Caxias – RJ

Mundo 3 Período 2023.2

Missão pratica nível 2

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

Objetivos 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.

Link no github: https://github.com/lucasxpietro/missaonv2.git

Código do 1° Procedimento:

```
CREATE DATABASE loja;

USE loja;

CREATE TABLE
	Pessoa (
		id_Pessoa INTEGER NOT NULL IDENTITY (1, 1) PRIMARY KEY,
		nome VARCHAR(255) NOT NULL,
		logradouro VARCHAR(255) NOT NULL,
		cidade VARCHAR(255) NOT NULL,
```

```
		estado CHAR(2) NOT NULL,
		telefone VARCHAR(11) NOT NULL,
		email VARCHAR(255) NOT NULL
	);

CREATE TABLE
	Produtos (
		idProdutos INTEGER NOT NULL IDENTITY (1, 1) PRIMARY KEY,
		nome VARCHAR(255) NOT NULL,
		quantidade INTEGER NOT NULL,
		precoVenda NUMERIC NOT NULL
	);

CREATE TABLE
	Usuarios (
		id_Usuario INTEGER NOT NULL IDENTITY (1, 1) PRIMARY KEY,
		login VARCHAR(50) NOT NULL,
		senha VARCHAR(50) NOT NULL
	);

CREATE TABLE
	Pessoa_Fisica (
		idPessoa_Fisica INTEGER NOT NULL PRIMARY KEY,
		cpf VARCHAR(11) NOT NULL,
		idPessoa INTEGER NULL,
		constraint fk_PessoaFisica_Pessoa foreign key (idPessoa) references
dbo.Pessoa (id_Pessoa)
	);

CREATE TABLE
	Pessoa_Juridica (
		idPessoa_Juridica INTEGER NOT NULL PRIMARY KEY,
		cnpj varchar(14) NOT NULL,
		idPessoa INTEGER NULL,
		constraint fk_PessoaJuridica_Pessoa foreign key (idPessoa) references
dbo.Pessoa (id_Pessoa)
	);

CREATE TABLE
	Movimento (
		idMovimento INTEGER NOT NULL IDENTITY (1, 1) PRIMARY KEY,
		idUsuario INTEGER NOT NULL,
		idPessoa INTEGER NOT NULL,
		idProduto INTEGER NOT NULL,
		quantidade INTEGER NOT NULL,
		tipo CHAR(1) NOT NULL,
		valorUnitario FLOAT NOT NULL,
		constraint fk_Movimento_Produto foreign key (idProduto) references
dbo.Produtos (idProdutos),
		constraint fk_Movimento_Usuario foreign key (idUsuario) references
dbo.Usuarios (id_Usuario),
		constraint fk_Movimento_Pessoa foreign key (idPessoa) references dbo.Pessoa
(id_Pessoa)
	);

CREATE SEQUENCE dbo.CodigoPessoa
    START WITH 1
    INCREMENT BY 1 ;
```

Análise e Conclusão:

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

1:1 (Um para Um): Cada registro em uma tabela está ligado a um único registro em outra tabela e vice-versa, geralmente usando chaves estrangeiras únicas.

1:N (Um para Muitos): Cada registro em uma tabela pode estar ligado a vários registros em outra tabela, enquanto cada registro na segunda tabela está ligado a apenas um registro na primeira, usando uma chave estrangeira.

N:N (Muitos para Muitos): Múltiplos registros em uma tabela podem se relacionar com múltiplos registros em outra, geralmente usando uma tabela intermediária com chaves estrangeiras que se conectam às tabelas principais.

Essas cardinalidades permitem representar e estruturar relacionamentos de dados de forma eficaz em um banco de dados relacional.

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

Para representar herança em bancos de dados relacionais, geralmente se utiliza o modelo de "herança de tabela" (table inheritance). Nesse modelo, cada classe ou entidade é representada por uma tabela no banco de dados, com a tabela da classe base contendo atributos comuns e as tabelas das subclasses contendo atributos específicos. Isso permite modelar a herança de classes, onde as subclasses herdam atributos da superclasse. Isso é comumente implementado em sistemas de gerenciamento de banco de dados que suportam herança, como o PostgreSQL.

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 da Microsoft que aprimora a produtividade no gerenciamento de bancos de dados SQL Server. Ele oferece uma interface amigável, permite escrever e executar consultas SQL, gerenciar tabelas, esquemas e segurança, automatizar tarefas, monitorar o desempenho e oferece integração com outras ferramentas. Isso torna mais eficiente o trabalho de administradores e desenvolvedores na administração e desenvolvimento de bancos de dados SQL Server.

Curso - Desenvolvimento full Stack Disciplina – Iniciando o caminho pelo Java

Lucas Pietro Santos de Souza Matricula – 2022 0809 0648

Campus - Polo Duque de Caxias – RJ

Mundo 3 Período 2023.2

Missão prática nível 2

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

Objetivos 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.

Link no github: https://github.com/lucasxpietro/missaonv2.git

Código do 2° Procedimento:

Alimentando a base:

```
use loja

insert
```

```
	into
	Usuarios(
	login,
	senha
	)
values
	(
	'op1',
	'op1'
	),
	(
	'op2',
	'op2'
	);

insert
	into
	Usuarios(
	login,
	senha
	)
values
	(
	'op3',
	'op3'
	),
	(
	'op4',
	'op4'
	);
select * FROM Usuarios;

insert
	into
	Produtos(
	nome,
	quantidade,
	precoVenda
	)
values
	(
	'banana',
	100,
	5.0
	),
	(
	'laranja',
	500,
	2.0
	),
	(
	'Manga',
	800,
	4.0
	);

insert
	into
```

```
	Produtos(
	nome,
	quantidade,
	precoVenda
	)
values
	(
	'uva',
	300,
	6.0
	),
	(
	'Maça',
	300,
	3.0
	);
select * from Produtos;

insert
	into
	Pessoa(
		nome,
		logradouro,
		cidade,
		estado,
		telefone,
		email
	)
values
	(
		'Lucas',
		'Rua mendes',
		'São Joao de Meriti',
		'RJ',
		'25585590',
		'raypietrosantos@gmail.com'
	);
select * from Pessoa;

select
	@@IDENTITY;



select
	*
from
	Pessoa;

select
	*
from
	Pessoa_Fisica;

select
	*
from
	Pessoa_Fisica
```

```
inner join Pessoa on
	Pessoa.id_Pessoa = Pessoa_Fisica.idPessoa;

select
	Pessoa_Fisica.cpf,
	Pessoa.nome
from
	Pessoa_Fisica
inner join Pessoa on
	Pessoa.id_Pessoa = Pessoa_Fisica.idPessoa;

insert
	into
	Pessoa_Fisica (idPessoa_Fisica,
	cpf,
	idPessoa)
values
	(NEXT VALUE FOR CodigoPessoa,
'00011122299',
1);

Select * from Pessoa_Fisica;

insert
	into
	Pessoa(
		nome,
		logradouro,
		cidade,
		estado,
		telefone,
		email
	)
values
	(
		'Lucas LTDA',
		'Parque Araruama',
		'São Joao de Meriti',
		'RJ',
		'25585590',
		'raypietrosantos@gmail.com'
	);
Select * from Pessoa;

insert
	into
	Pessoa_Juridica(idPessoa_Juridica,
	cnpj,
	idPessoa)
values
	(NEXT VALUE FOR CodigoPessoa,
'00111222000199',
2);

Select * from Pessoa_Juridica;

select
	*
```

```
from
	Pessoa_Juridica
inner join Pessoa on
	Pessoa.id_Pessoa = Pessoa_Juridica.idPessoa;

insert
	into
	movimento(
		idUsuario,
		idPessoa,
		idProduto,
		quantidade,
		tipo,
		valorUnitario
	)
values
	(
	1,
	1,
	1,
	10,
	'E',
	3.0
	),
	(
	1,
	1,
	2,
	10,
	'E',
	1.0
	),
	(
	1,
	1,
	3,
	10,
	'E',
	3.0
	);

insert
	into
	movimento(
		idUsuario,
		idPessoa,
		idProduto,
		quantidade,
		tipo,
		valorUnitario
	)
values
	(
	2,
	2,
	1,
	3,
	'S',
```

```
        5.0
        ),
        (
        2,
        2,
        2,
        1,
        'S',
        2.0
        ),
        (
        2,
        2,
        3,
        8,
        'S',
        4.0
        );

insert
        into
        movimento(
                idUsuario,
                idPessoa,
                idProduto,
                quantidade,
                tipo,
                valorUnitario
        )
values
        (
        3,
        2,
        1,
        5,
        'E',
        4.75
        ),
        (
        3,
        2,
        2,
        7,
        'E',
        1.32
        ),
        (
        3,
        1,
        1,
        9,
        'S',
        5.0
        ),
        (
        3,
        1,
        2,
```

```
        3,
        'S',
        2.0
        ),
        (
        4,
        2,
        1,
        8,
        'S',
        5
        ),
        (
        4,
        2,
        2,
        6,
        'S',
        2
        );

select
        *
from
        Movimento
order by
        tipo;

select
        *
from
        Pessoa;
```

Consultas SQL:

```
use loja
-- 4. a) Dados completos de pessoas físicas.

select * from Usuarios

select
        Pessoa_Fisica.idPessoa_Fisica as id,
        Pessoa_Fisica.cpf,
        p.nome,
        p.logradouro,
        p.cidade,
        p.estado,
        p.telefone,
        p.email
from Pessoa_Fisica
INNER JOIN Pessoa as p on Pessoa_Fisica.idPessoa = p.id_Pessoa;

-- 4. b) Dados completos de pessoas jurídicas.
```

```sql
select
	Pessoa_Juridica.idPessoa_Juridica as id,
	Pessoa_Juridica.cnpj,
	p.nome,
	p.logradouro,
	p.cidade,
	p.estado,
	p.telefone,
	p.email
	from Pessoa_Juridica
INNER JOIN Pessoa as p on Pessoa_Juridica.idPessoa = p.id_Pessoa;

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

select
	Produtos.idProdutos,
	Produtos.nome,
	Pessoa.nome as 'Fornecedor',
	Movimento.quantidade,
	Movimento.valorUnitario as 'PrecoUnitario',
	Movimento.quantidade * Movimento.valorUnitario as 'ValorTotal'
	from Movimento
INNER JOIN Produtos on idProduto = Produtos.idProdutos
INNER JOIN Pessoa on Movimento.idPessoa = Pessoa.id_Pessoa
where Movimento.tipo = 'E'

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

select
	Produtos.idProdutos,
	Produtos.nome,
	Pessoa.nome,
	Movimento.idMovimento,
	Movimento.quantidade,
	Movimento.valorUnitario as 'PrecoUnitario',
	Movimento.quantidade * Movimento.valorUnitario as 'ValorTotal'
from
	Movimento
INNER JOIN Produtos on Movimento.idProduto = Produtos.idProdutos
INNER JOIN Pessoa on Movimento.idPessoa = Pessoa.id_Pessoa
where
	Movimento.tipo = 'S'

-- 4. e) Valor total das entradas agrupadas por produto.

select
	Movimento.idProduto as id,
	Produtos.nome,
	sum (Movimento.valorUnitario) * sum (Movimento.quantidade) as
'ValorTotalporProduto'
from
	Movimento
INNER JOIN Produtos on Produtos.idProdutos = Movimento.idProduto

where
```

```
	Movimento.tipo = 'E'
group by
	Movimento.idProduto,
	Produtos.nome

-- 4. f) Valor total das saídas agrupadas por produto.

select
	Movimento.idProduto as id,
	Produtos.nome,
	sum(Movimento.valorUnitario) * sum (Movimento.quantidade) as
'ValorTotalPorProduto'
from
	Movimento
INNER JOIN Produtos on Produtos.idProdutos = Movimento.idProduto
where
	Movimento.tipo = 'S'
group by
	Movimento.idProduto,
	Produtos.nome

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


select
	Usuarios.id_Usuario,
	Usuarios.login
from
	Usuarios where id_Usuario not in (
		select
			distinct Movimento.idUsuario
		from
			Movimento
		where
			Movimento.tipo = 'E'
	)

-- 4. h) Valor total de entrada, agrupado por operador.

select
	Usuarios.login,
	sum (Movimento.valorUnitario) * sum (Movimento.quantidade) as
'ValorTotalporProduto'
from
	Movimento
INNER JOIN Usuarios on Movimento.idUsuario = Usuarios.id_Usuario
where
	Movimento.tipo = 'E'
group by
	Movimento.idUsuario,
	Usuarios.login

-- 4. i) Valor total de saída, agrupado por operador.

select
	Usuarios.login,
	sum (Movimento.valorUnitario) * sum (Movimento.quantidade) as
'ValorTotalporProduto'
```

```sql
from
	Movimento
INNER JOIN Usuarios on Movimento.idUsuario = Usuarios.id_Usuario
where
	Movimento.tipo = 'S'
group by
	Movimento.idUsuario,
	Usuarios.login

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

select
	Movimento.idProduto,
	Produtos.nome,
	(sum (Movimento.valorUnitario) * sum (Movimento.quantidade)) / COUNT
(Movimento.idMovimento) as 'ValorMedioDeVenda'
from
	Movimento
INNER JOIN Produtos on Movimento.idProduto = Produtos.idProdutos
group by
	Movimento.idProduto,
	Produtos.nome
```

Análise e Conclusão

Quais as diferenças no uso de sequence e identity?

A principal diferença é que as "sequences" são mais flexíveis e podem ser usadas em sistemas de gerenciamento de banco de dados que suportam esse recurso, enquanto o "identity" é específico do SQL Server e é mais simples, mas menos flexível. A escolha entre eles depende dos requisitos específicos do seu sistema e do sistema de gerenciamento de banco de dados que você está usando.

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

As chaves estrangeiras são um mecanismo crucial para garantir a integridade dos dados em um banco de dados relacional, prevenindo inserções incorretas, atualizações inconsistentes e exclusões inadequadas. Elas são uma parte fundamental das práticas recomendadas de modelagem de dados e administração de bancos de dados.

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

No SQL, alguns operadores pertencem à álgebra relacional, como SELECT, FROM, WHERE e JOIN, que são usados para recuperar e manipular dados em tabelas. Outros operadores, como IN, EXISTS, UNION, INTERSECT, EXCEPT e ALL, são definidos no cálculo relacional e são usados para expressar consultas complexas e condicionais. O SQL combina elementos desses modelos teóricos para fornecer uma linguagem poderosa para consultar bancos de dados relacionais, oferecendo flexibilidade e expressividade.

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

O agrupamento em consultas SQL é feito usando a cláusula GROUP BY. O requisito obrigatório é especificar as colunas pelas quais você deseja agrupar os resultados. Além disso, ao usar o GROUP BY, todas as colunas na lista de seleção devem estar presentes na cláusula GROUP BY ou ser funções de agregação. Isso permite calcular valores resumidos (agregados) para cada grupo de dados com base nas colunas de agrupamento. O uso correto do GROUP BY é fundamental para obter resultados precisos em consultas que envolvem agregação de dados.