UNIVERSIDADE ESTÁCIO DE SÁ

FULLSTACK

Mundo 03 - Nível 02

Implementação de um cadastro de clientes em modo texto, com persistência em arquivos, baseado na tecnologia Java.

Herval Rosano Dantas

Matrícula 202205119203

RIO DE JANEIRO – RJ

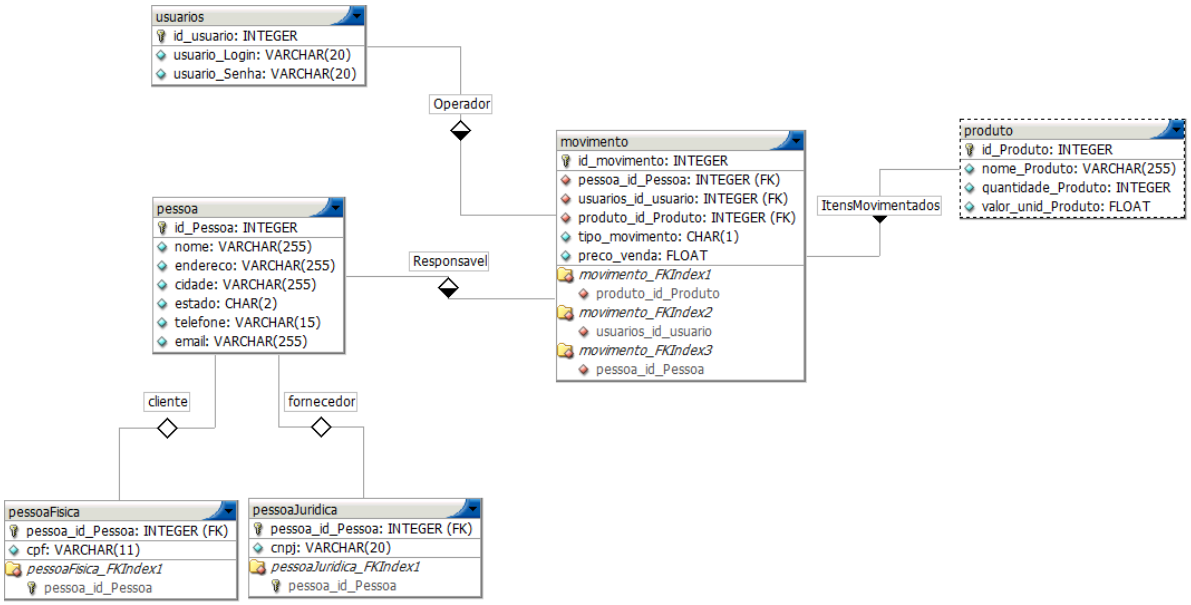
2023

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).

1º Procedimento – Criando o Banco de Dados

Modelagem do Banco de Dados usando o DBDesigner Fork



Com a ajuda do DBDesigner, passamos a criar as tabelas da loja. Veja arquivo loja_create_tables.sql

```sql
-- =============== Criando Tabela Usuário

CREATE TABLE [Usuario] (
  idUsuario INT IDENTITY(1,1) PRIMARY KEY,
  Login VARCHAR(20) NULL,
  Senha VARCHAR(20) NULL
);

-- =============== Inserindo dados na tabela Usuario
INSERT INTO [Usuario] (Login, Senha)
VALUES ('op1', 'op1'), ('op2', 'op2');

SELECT * FROM [Usuario];
```

100 %

Results | Messages

	idUsuario	Login	Senha
1	1	op1	op1
2	2	op2	op2

```sql
-- =============== Criando Tabela Produto
CREATE TABLE [Produto] (
  idProduto INT IDENTITY(1,1) NOT NULL,
  nome VARCHAR(255) NULL,
  quantidade INT NULL,
  precoVenda FLOAT,
  PRIMARY KEY (idProduto)
);
-- =============== Inserindo dados na tabela Produto
INSERT INTO [Produto] (nome, quantidade, precoVenda)
VALUES ('Banana', 200, 7.30),
       ('Manga', 300, 8.90),
       ('Laranja', 600, 5.50),
       ('Maçã', 200, 3.00),
       ('Jaca', 50, 10.00),
       ('Goiaba', 400, 3.50),
       ('Pitomba', 100, 2.00);

SELECT * FROM [Produto];
```

00 %

Results | Messages

	idProduto	nome	quantidade	precoVenda
1	1	Banana	200	7,3
2	2	Manga	300	8,9
3	3	Laranja	600	5,5
4	4	Maçã	200	3
5	5	Jaca	50	10
6	6	Goiaba	400	3,5
7	7	Pitomba	100	2

```sql
-- =============== Tabela Pessoa
CREATE TABLE [Pessoa] (
  idPessoa INT IDENTITY(1,1) NOT NULL,
  nome VARCHAR(255) NULL,
  logradouro VARCHAR(255) NULL,
  cidade VARCHAR(255) NULL,
  estado CHAR(2) NULL,
  telefone VARCHAR(11) NULL,
  email VARCHAR(255) NULL,
  PRIMARY KEY (idPessoa)
);

-- =============== Inserindo dados na tabela Pessoa
INSERT INTO [Pessoa] (nome, logradouro, cidade, estado, telefone, email)
VALUES ('Herval Rosano Dantas', 'Rua Ituverava 866', 'Rio de Janeiro', 'RJ', '21991870000', 'hervaldantas@gmail.com'),
       ('Angela Agostinho da C. Dantas', 'Rua Araguaia 111', 'Rio de Janeiro', 'RJ', '11133328909', 'angeladantas@gmail.com'),
       ('Antonio Dantas', 'Rua Tirol 777', 'Rio de Janeiro', 'RJ', '11991098789', 'antoniodantas@gmail.com'),
       ('Renato Dantas', 'Rua México', 'Rio de Janeiro', 'RJ', '11909872222', 'renatodantas@gmail.com');
```

100 %

Results | Messages

	idPessoa	nome	logradouro	cidade	estado	telefone	email
1	1	Herval Rosano Dantas	Rua Ituverava 866	Rio de Janeiro	RJ	21991870000	hervaldantas@gmail.com
2	2	Angela Agostinho da C. Dantas	Rua Araguaia 111	Rio de Janeiro	RJ	11133328909	angeladantas@gmail.com
3	3	Antonio Dantas	Rua Tirol 777	Rio de Janeiro	RJ	11991098789	antoniodantas@gmail.com
4	4	Renato Dantas	Rua México	Rio de Janeiro	RJ	11909872222	renatodantas@gmail.com

```sql
-- =============== Criando Tabela PessoaFisica
CREATE TABLE [PessoaFisica] (
  idPessoaFisica INT IDENTITY(1,1) NOT NULL,
  idPessoa INT NOT NULL,
  nome VARCHAR(255) NULL,
  logradouro VARCHAR(255) NULL,
  cidade VARCHAR(255) NULL,
  estado CHAR(2) NULL,
  telefone VARCHAR(11) NULL,
  email VARCHAR(255) NULL,
  CPF VARCHAR(14) NULL,
  PRIMARY KEY(idPessoaFisica),
  CONSTRAINT FK_PessoaFisica_Pessoa
  FOREIGN KEY (idPessoa)
  REFERENCES [Pessoa] (idPessoa)
);

 -- =============== Inserindo Tabela PessoaFisica
INSERT INTO [PessoaFisica] (idPessoa, nome, logradouro, cidade, estado, telefone, email, CPF)
VALUES (3, 'Carlos Ronan Dantas', 'Rua Jose Venâncio 596', 'Rio de Janeiro', 'RJ', '12345678903', 'calosronan@gmail.com',  '0122094005-40'),
       (4, 'José Ronilson', 'Rua Cascavel  2', 'Rio de Janeiro', 'RJ', '12345678904', 'ronsilsondantas2@gmail.com', '022055785-50');

SELECT * FROM [PessoaFisica];
```

00 %

Results | Messages

	idPessoaFisica	idPessoa	nome	logradouro	cidade	estado	telefone	email	CPF
1	1	3	Carlos Ronan Dantas	Rua Jose Venâncio 596	Rio de Janeiro	RJ	12345678903	calosronan@gmail.com	0122094005-40
2	2	4	José Ronilson	Rua Cascavel 2	Rio de Janeiro	RJ	12345678904	ronsilsondantas2@gmail.com	022055785-50

```sql
-- =============== Criando Tabela PessoaJuridica
CREATE TABLE [PessoaJuridica] (
  idPessoaJuridica INT IDENTITY(1,1) NOT NULL,
  idPessoa INT NOT NULL,
  nome VARCHAR(255) NULL,
  logradouro VARCHAR(255) NULL,
  cidade VARCHAR(255) NULL,
  estado CHAR(2) NULL,
  telefone VARCHAR(11) NULL,
  email VARCHAR(255) NULL,
  CNPJ VARCHAR(14) NULL,
  PRIMARY KEY (idPessoaJuridica),
  CONSTRAINT FK_PessoaJuridica_Pessoa
  FOREIGN KEY (idPessoa)
  REFERENCES [Pessoa] (idPessoa)
);

-- =============== Inserindo dados na tabela PessoaJuridica
INSERT INTO [PessoaJuridica] (idPessoa, nome, logradouro, cidade, estado, telefone, email, CNPJ)
VALUES (1, 'Digital Minda', 'Rua Ituverava 21', 'Rio de Janeiro', 'RJ', '1111234242', 'digitalmind1@gmail.com', '11163258000146'),
       (2, 'H2A Design', 'Rua Tirol 2', 'Rio de Janeiro', 'RJ', '12345678902', 'h2adesign@gmail.com', '18263258000146');
```

00 %

Results | Messages

	idPessoaJuridica	idPessoa	nome	logradouro	cidade	estado	telefone	email	CNPJ
1	1	1	Digital Minda	Rua Ituverava 21	Rio de Janeiro	RJ	1111234242	digitalmind1@gmail.com	11163258000146
2	2	2	H2A Design	Rua Tirol 2	Rio de Janeiro	RJ	12345678902	h2adesign@gmail.com	18263258000146

```sql
-- ================ Criando Tabela Movimentação
CREATE TABLE [Movimentacao] (
  id_movimento INT IDENTITY(1,1) NOT NULL,
  idUsuario INT NULL,
  idPessoa INT NULL,
  idProduto INT NULL,
  quantidade INT NOT NULL,
  tipo CHAR(1) NULL,
  valor_unitario FLOAT NULL,
  CONSTRAINT PK_Movimentacao PRIMARY KEY (id_movimento),
  CONSTRAINT FK_Usuario FOREIGN KEY (idUsuario)
    REFERENCES [Usuario] (idUsuario),
  CONSTRAINT FK_Pessoa FOREIGN KEY (idPessoa)
    REFERENCES [Pessoa] (idPessoa),
  CONSTRAINT FK_Produto FOREIGN KEY (idProduto)
    REFERENCES [Produto] (idProduto)
);

-- Inserindo dados na tabela Movimentacao
INSERT INTO [Movimentacao] (idUsuario, idPessoa, idProduto, quantidade, tipo, valor_unitario)
VALUES (1, 1, 1, 150, 'E', 8.90),
       (2, 2, 2, 150, 'S', 8.90);
```

00 %

Results | Messages

	id_movimento	idUsuario	idPessoa	idProduto	quantidade	tipo	valor_unitario
1	1	1	1	1	150	E	8,9
2	2	2	2	2	150	S	8,9

Conforme solicitado pela missão. Vamos efetuar diversas pesquisas. Veja arquivo loja_searches.sql.

Os dados completos de pessoas físicas e pessoas jurídicas já foram apresentados acima quando da criação dos mesmos.

```sql
-- 1- Movimentações de entrada, com produto, fornecedor, quantidade, preço unitário e valor total.
SELECT p.nome, m.idProduto, m.idPessoa, m.quantidade, m.valor_unitario, (m.quantidade * m.valor_unitario) AS valor_total
FROM Movimentacao AS m
JOIN Pessoa AS p ON m.idPessoa = p.idPessoa
WHERE m.tipo = 'E';
```

00 %

Results | Messages

	nome	idProduto	idPessoa	quantidade	valor_unitario	valor_total
1	Herval Rosano Dantas	1	1	150	8,9	1335
2	Antonio Dantas	2	3	250	10,22	2555
3	Rosângela Dantas	4	5	380	4,9	1862
4	Rosângela Dantas	4	5	200	6,9	1380
5	Rosânia Cristina Dantas	1	7	450	4,44	1998

```sql
-- 2- Movimentações de saída, com produto, cliente, quantidade, preço unitário e valor total.

SELECT p.nome, m.idProduto, m.idPessoa, m.quantidade, m.valor_unitario, (m.quantidade * m.valor_unitario) AS valor_total
FROM Movimentacao AS m
JOIN Pessoa AS p ON m.idPessoa = p.idPessoa
WHERE m.tipo = 'S';
```

100 %

Results | Messages

	nome	idProduto	idPessoa	quantidade	valor_unitario	valor_total
1	Angela Agostinho da C. Dantas	2	2	300	20,51	6153
2	Antonio Dantas	3	3	520	6,4	3328
3	Renato Dantas	7	4	100	3,5	350
4	Renato Dantas	5	4	80	7,33	586,4
5	Francisco Rafael Dantas	2	9	550	5,55	3052,5

```sql
-- 3- Valor total das entradas agrupadas por produto.
SELECT idProduto, SUM(quantidade * valor_unitario) AS ValorTotalEntrada
FROM Movimentacao
WHERE tipo = 'E'
GROUP BY idProduto;
```

100 %

Results | Messages

	idProduto	ValorTotalEntrada
1	1	3333
2	2	2555
3	4	3242

```sql
-- 4- Valor total das saídas agrupadas por produto.
SELECT idProduto, SUM(quantidade * valor_unitario) AS ValorTotalSaida
FROM Movimentacao
WHERE tipo = 'S'
GROUP BY idProduto;
```

100 %

Results | Messages

	idProduto	ValorTotalSaida
1	2	9205,5
2	3	3328
3	5	586,4
4	7	350

```sql
-- 5-  Operadores que não efetuaram movimentações de entrada (compra).
SELECT *
FROM Usuario
WHERE idUsuario NOT IN (SELECT DISTINCT idUsuario FROM Movimentacao WHERE tipo = 'E');
```

100 %

Results | Messages

idUsuario	Login	Senha

```sql
-- 6- Valor total de entrada, agrupado por operador.
SELECT m.idUsuario, u.login, SUM(m.quantidade * m.valor_unitario) AS ValorTotalEntrada
FROM Movimentacao m
JOIN Usuario u ON m.idUsuario = u.idUsuario
WHERE m.tipo = 'E'
GROUP BY m.idUsuario, u.login;
```

00 %

Results | Messages

	idUsuario	login	ValorTotalEntrada
1	1	op1	5270
2	2	op2	3860

```sql
-- 7- Valor total de saída, agrupado por operador.F
SELECT p.nome AS operador, SUM(m.quantidade * m.valor_unitario) AS valor_total_saida
FROM Movimentacao AS m
JOIN Pessoa AS p ON m.idPessoa = p.idPessoa
WHERE m.tipo = 'S'
GROUP BY p.nome;
```

100 %

Results | Messages

	operador	valor_total_saida
1	Angela Agostinho da C. Dantas	6153
2	Antonio Dantas	3328
3	Francisco Rafael Dantas	3052,5
4	Renato Dantas	936,4

```sql
---8-  Valor médio de venda por produto, utilizando média ponderada.
SELECT idProduto, SUM(quantidade * valor_unitario) / SUM(quantidade) AS ValorMedioVenda
FROM Movimentacao
WHERE tipo = 'S'
GROUP BY idProduto;
```

Invalid column name 'valor_unitario'.

100 %

Results | Messages

	idProduto	ValorMedioVenda
1	2	10,83
2	3	6,4
3	5	7,33
4	7	3,5

Análise e Conclusão:

Quais as diferenças no uso de sequence e identity?

Ambos são geradores de sequência numérica num Banco de dado. A diferença é que identity é criado na coluna da própria tabela onde está sendo declarada. Já o sequence é um objeto que não está vinculado a uma tabela específica e que por isso pode ser usada em mais de uma.

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

As chaves estrangeiras (foreign Keys – FK) é justamente o faz com que haja relação entre um dado em uma row (linha) de uma tabela com a row na outra tabela. É essa linkagem que mantém a integridade das informações, por isso que chamamos de Bando de Dados Relacional. Nesse relacionamento é que se fortalece a integridade das informações, nos permite fazer as junções (Join), integridade referencial, prevenção de dados inválidos, e muito mais.

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

Os operadores SQL que pertencem à álgebra relacional: seleção, projeção, união, interseção e diferença.
Seleção, é quando se recupera uma linha de uma tabela que atenda a uma condição específica do tipo: coluna idade > 30.
Projeção, é usado para selecionar colunas específicas de uma tabela e com elas criar uma nova tabela.
União, é usado para combinar duas tabelas e retorna todas as linhas distintas.
Interseção, é usado para encontrar as linhas que estão em ambas as tabelas.
Diferença, é quando retorna as linhas que estão numa tabela mas não na outra.

Já o cálculo relacional fica mais próximo da lógica matemática e pode ser mais abstrato do que a álgebra relacional, que é mais operacional. Na linguagem SQL, os operadores como SELECT, FROM, WHERE, JOIN, GROUP BY, HAVING, ORDER BY, entre outros, são usados para realizar consultas e manipulações de dados. Eles representam operações mais práticas e funcionais que são usadas em bancos de dados do mundo real.

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

O agrupamento é feito agrupando-se linhas (row) de dados com valores comuns em mais de uma coluna (column) onde se aplica uma função de ajuntamento como SUM, COUNT, AVG, MAX OU MIN a esses grupos resultantes. E claro que o requisito obrigatório é GROUP BY.

Um exemplo básico e padrão para um agrupamento:

```
SELECT coluna1, coluna2, funcao_agregacao(coluna3) // aqui na coluna3 pode ser aplicada
                                                   uma função de agregação do tipo: um
                                                   somatório, uma média, valores a partir
                                                   de um certo número e por aí vai.
FROM tabela
GROUP BY coluna1, coluna2;
```

SELECT: Especifica as colunas que você deseja selecionar na consulta.
FROM: Especifica a tabela da qual você está selecionando os dados.
GROUP BY: Indica as colunas que servirão como critério de agrupamento. As linhas de dados serão agrupadas com base nos valores dessas colunas.

Atenção: apenas as colunas listadas na cláusula GROUP BY podem ser selecionadas na lista de seleção (SELECT). Uma coisa que observei no SQL é que a ordem das variáveis pode vir antes de serem criadas. Exemplo. Está se selecionando colunas antes de dizer em qual tabela elas estão e além disso qual a restrição “GROUP BY”, digo: só se pode usar as colunas que estão especificadas no agrupamento.