

Relatório Projeto Final

Projeto: João Atacadão Rebirth

Lucas de Almeida Bandeira Macedo – 19/0047089

João Pedro Felix de Almeida – 19/0015292

Introdução

O projeto consiste na implementação de um banco de dados para uma loja física (e fictícia) de departamentos. A ideia surgiu quando, no semestre passado, desenvolvemos o projeto final da disciplina “Técnicas de Programação 1”, momento que desenvolvemos todo o aplicativo de gestão da empresa, que inclui: cadastro de funcionários, produtos, clientes e controle de caixa, o que consequentemente gerou a necessidade de criação, leitura, atualização e remoção de dados. Na época, devido à falta de conhecimento de banco de dados, a implementação foi realizada com o uso de arquivos texto, para simular um banco de dados profissional. Com isso em mente, optamos por nesse semestre concluir a implementação do projeto com um banco de dados nos padrões comerciais.

Para isso, foi necessária a criação de um novo pacote de códigos-fonte que atuam como a camada de persistência da aplicação. Além disso, para fazer melhor uso do banco de dados, também foi realizada a criação de uma nova tela capaz de manipular o histórico de compras realizadas na loja com o uso opcional de filtros.

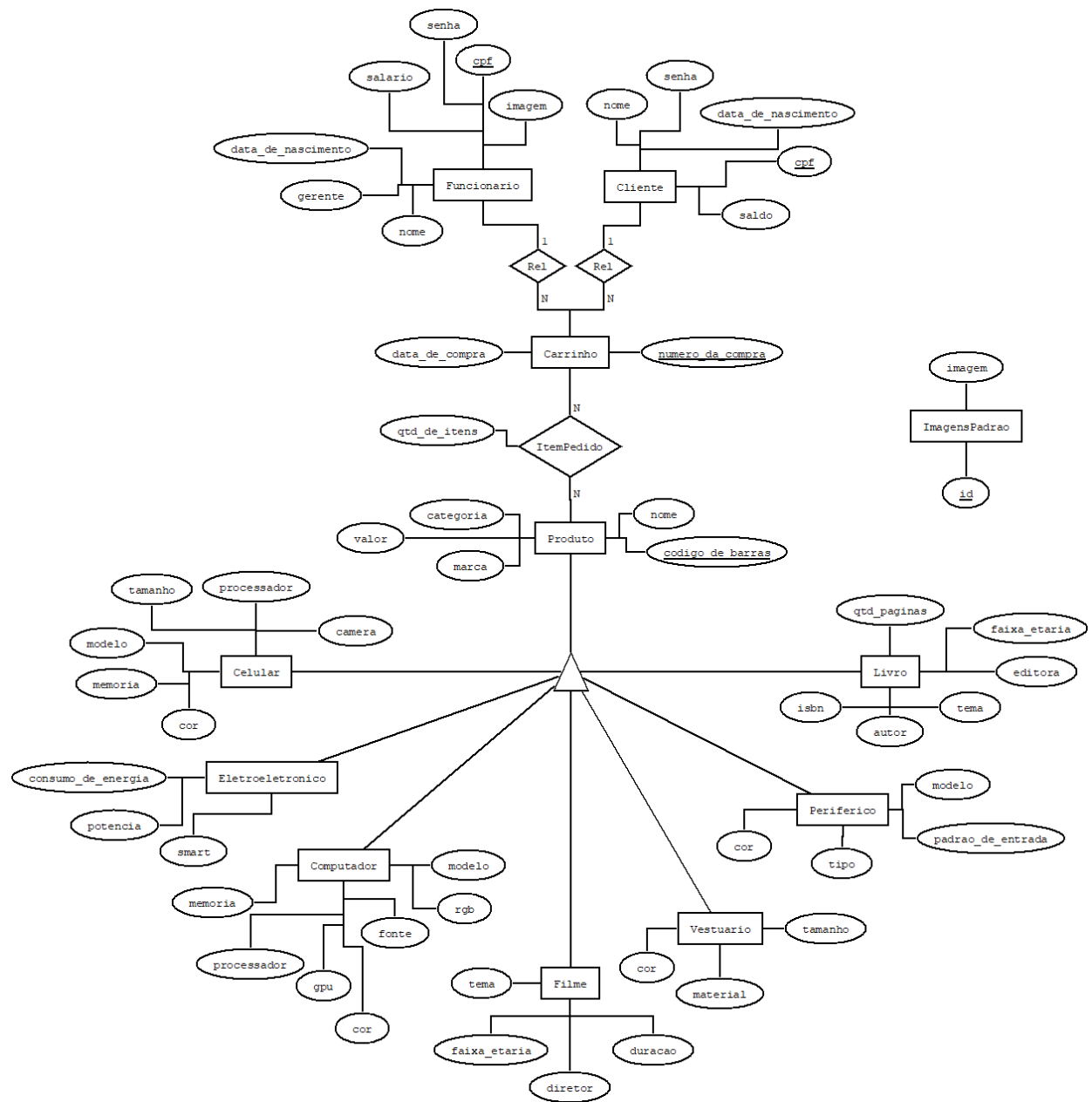
A implementação se baseia nas seguintes tabelas:

- ◆ Produto: da qual derivam outras sete: celular, eletroeletrônico, computador, filme, vestuário, periférico e livro.
- ◆ Cliente.
- ◆ Funcionário: que pode ou não ser um gerente.
- ◆ Carrinho
- ◆ Imagens Padrão: responsável por armazenar imagens para o caso de o funcionário não possuir uma foto.
- ◆ Item Pedido: que na realidade se trata do relacionamento entre as tabelas produto e carrinho.

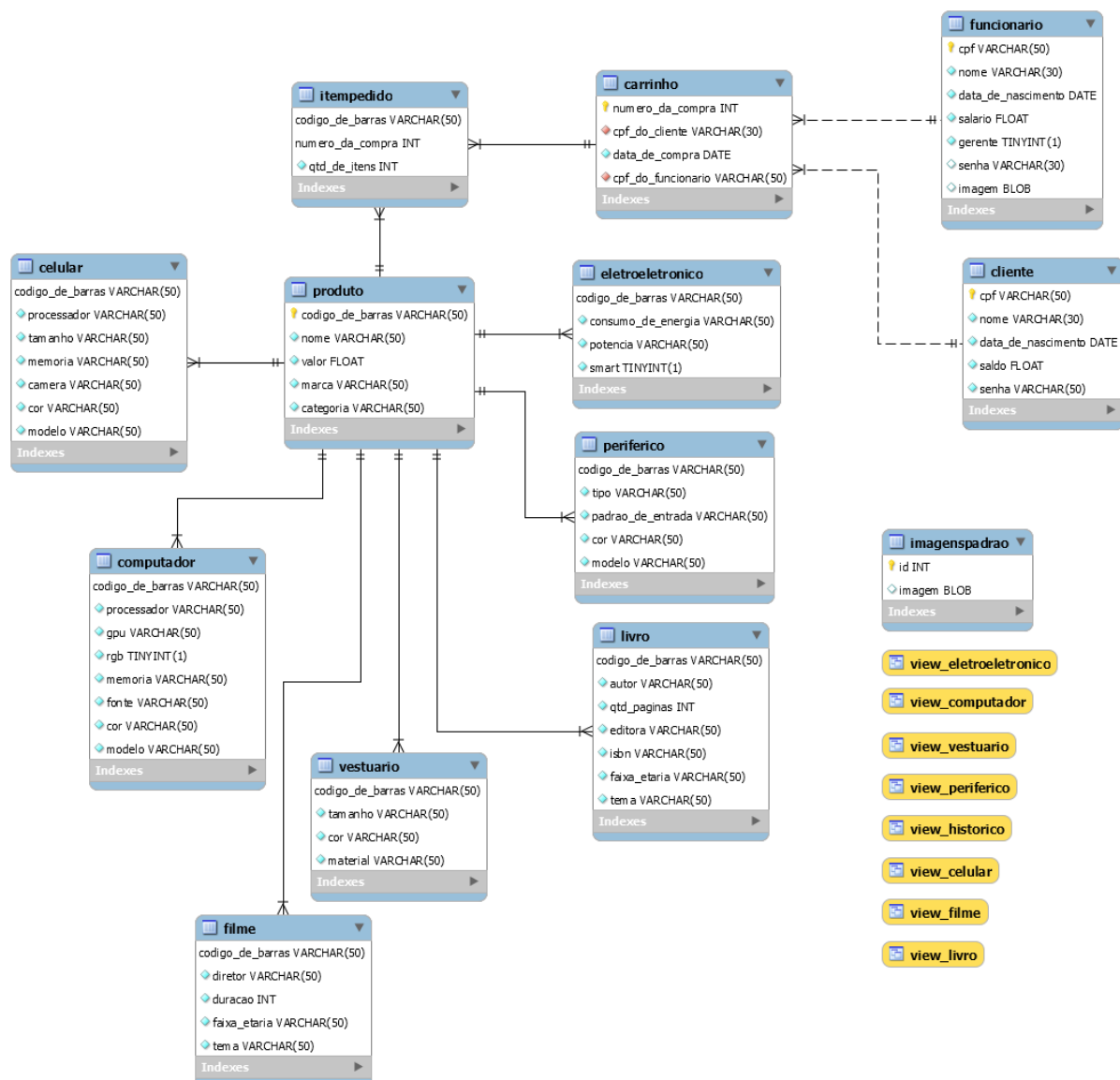
Ao longo da aplicação estão diversas chamadas a classe `Conexao.java` que, como o nome sugere, é responsável por realizar a conexão entre o banco de dados e a aplicação. Essas chamadas passam parâmetros aos métodos (estáticos) da classe `Conexao.java` que a partir deles realizam as operações clássicas de um CRUD: Criação (*Create*), Leitura (*Read*), Atualização (*Update*) e Remoção (*Delete*).

Nas próximas páginas se encontram os modelos entidade relacionamento e relacional que serviram como base para a criação do projeto, bem como a exemplificação de cinco consultas em álgebra relacional que poderiam ser realizadas sobre o banco de dados desenvolvido para a aplicação. Por fim, são apresentadas provas de normalização para 5 tabelas do banco de dados. No final do relatório é possível encontrar um diagrama que representa o relacionamento entre a interface gráfica da aplicação, a classe `Conexao.java` e o banco de dados.

Modelo Entidade Relacionamento



Modelo Relacional



Álgebra Relacional

1) A equação abaixo realiza um produto cartesiano entre as tabelas *ItemPedido*, *Carrinho* e *Produto*. O resultado por sua vez nada mais é que todas as colunas da tabela *ItemPedido*, seguidas pelas colunas da tabela *Carrinho* e *Produto*

$$\sigma_{ItemPedido.numero_da_compra=Carrinho.numero_da_compra \text{ and } ItemPedido.codigo_de_barra=Produto.codigo_de_barra}(ItemPedido \times Carrinho \times Produto)$$

2) A consulta abaixo realiza a projeção dos atributos nome das tabelas *Funcionário* e *Cliente* e número da compra da tabela *carrinho*. Note que essa projeção ocorre sobre o produto cartesiano das tabelas *Cliente*, *Funcionário* e *Carrinho*. Basicamente, estamos mostrando qual é o nome do funcionário que

operou uma compra, (e a compra também possui um número que é exibido,) para um cliente cujo nome também é exibido.

$$\pi_{\text{cliente.nome, funcionario.nome, carrinho.numero_da_compra}}(\sigma_{\text{carrinho.cpf_do_funcionario=funcionario.cpf and carrinho.cpf_do_cliente=cliente.cpf}}(\text{cliente} \times \text{funcionario} \times \text{carrinho}))$$

3) A diferença entre a consulta acima e a consulta abaixo é sutil, ainda sim, extremamente interessante. Aqui, estamos mostrando as compras dos funcionários que também são clientes da loja. Basicamente, o que ocorre é uma projeção do *nome* e *número da compra* desse *cliente* (e *funcionário*). Essa projeção é realizada sobre o produto cartesiano das tabelas *Cliente*, *Funcionário* e *Carrinho*.

$$\pi_{\text{Cliente.nome, Carrinho.numero_da_compra}}(\sigma_{\text{Cliente.cpf=Funcionario.cpf and Carrinho.cpf_do_cliente=Cliente.cpf}}(\text{Carrinho} \times \text{Cliente} \times \text{Funcionario}))$$

4) A consulta abaixo realiza a projeção dos atributos *nome*, *codigo_de_barras* e *numero_da_compra* da junção natural das tabelas *ItemPedido* e *Carrinho*, cujo resultado é tratado como a tabela *t* e, posteriormente, é realizada a junção da tabela *t* obtida anteriormente com a tabela *cliente* a partir de seu atributo *cpf*.

$$\pi_{\text{nome, codigo_de_barras, numero_da_compra}}(\rho_t(\text{ItemPedido} * \text{Carrinho}) \bowtie t.cpf_do_cliente=cliente.cpf(\text{Cliente}))$$

5) A consulta abaixo é um pouco mais complexa, sua análise será realizada de dentro para fora.

A) inicialmente realizamos uma junção entre as tabelas *ItemPedido* e *Produto* a partir de seus atributos *codigo_de_barras*. O resultado vai para a tabela *t*.

B) Em seguida, é feita uma projeção sobre as colunas *numero_da_compra* e *nome* obtidas a partir da tabela *t*. Naturalmente, essa projeção também uma tabela, o que nos permite ir para o passo C.

C) É feita uma junção com a tabela obtida a partir da projeção na etapa anterior e a tabela *Carrinho* a partir de seus atributos *numero_da_compra*. O resultado dessa junção vai para a tabela *s*.

D) Ao término do item C, é feita uma projeção sobre os atributos do *nome*, *cpf_do_cliente* e *numero_da_compra* da tabela *s* obtida a partir dos itens anteriores.

Basicamente, estamos consultando o nome de um produto, o número da compra em que esse produto foi comprado e o *cpf* do cliente que o comprou.

$$\pi_{s.nome, s.cpf_do_cliente, s.numero_da_compra} (s \leftarrow (\pi_{t.numero_da_compra, t.nome} (t \leftarrow ItemPedido \bowtie_{ItemPedido.codigo_de_barras=Produto.codigo_de_barras} Produto))) \bowtie_{Carrinho.numero_da_compra=t.numero_da_compra} Carrinho)$$

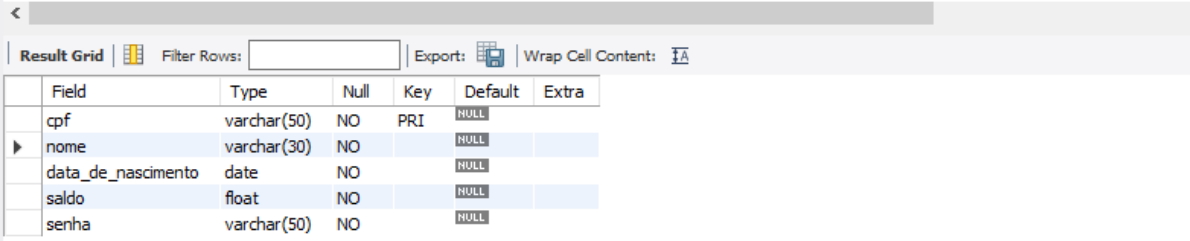
Análise das Formas Normais

1) Tabela Cliente

```

102 • CREATE TABLE IF NOT EXISTS Cliente
103 (
104     cpf                VARCHAR(50) PRIMARY KEY NOT NULL,
105     nome               VARCHAR(30) NOT NULL,
106     data_de_nascimento DATE NOT NULL,
107     saldo              FLOAT NOT NULL,
108     senha              VARCHAR(50) NOT NULL
109 );

```



Field	Type	Null	Key	Default	Extra
cpf	varchar(50)	NO	PRI	NULL	
nome	varchar(30)	NO		NULL	
data_de_nascimento	date	NO		NULL	
saldo	float	NO		NULL	
senha	varchar(50)	NO		NULL	

1FN) Como podemos ver, todos os atributos da tabela cliente são atômicos, afinal, não faz sentido que um cliente possua mais de um CPF, nome, data de nascimento, saldo ou senha. Além disso, cada um desses atributos pode ser expresso por um único valor.

2FN) Como visto, essa tabela está na forma normal 1 e então cumpre este requisito mínimo para estar na forma normal 2. Além disso, percebemos que todos os atributos do complemento da chave primária (no caso o CPF), isto é, nome, data de nascimento, saldo e senha são totalmente funcionalmente dependentes de CPF.

3FN) Como visto, essa tabela está na forma normal 2, e então cumpre este requisito mínimo para estar na forma normal 3. Além disso, percebemos que todos os atributos não-chave são dependentes não transitivos da chave primária pois:

- Podem existir pessoas com mesmo nome, mas nada garante que suas datas de nascimento, saldos ou senhas sejam iguais.
- Podem existir pessoas com mesma data de nascimento, mas nada garante que seus nomes, senha ou data de nascimento sejam iguais.
- Podem existir pessoas com mesmo saldo, mas nada garante que seus nomes, data de nascimento ou senha sejam iguais.
- Podem existir pessoas com mesma senha, mas nada garante que seus nomes, data de nascimento ou saldo sejam iguais.

Assim, como não há garantias de que um atributo não-chave possa definir outro atributo não-chave, não existe dependência transitiva entre eles. Logo, esta tabela está na terceira forma normal.

2) Tabela *ItemPedido*

```
136 • CREATE TABLE IF NOT EXISTS ItemPedido
137 (
138     codigo_de_barras    VARCHAR(50) NOT NULL,
139     numero_da_compra    INT NOT NULL,
140     qtd_de_itens        INT NOT NULL,
141
142     FOREIGN KEY(codigo_de_barras) REFERENCES Produto(codigo_de_barras),
143     FOREIGN KEY(numero_da_compra) REFERENCES Carrinho(numero_da_compra),
144
145     PRIMARY KEY(codigo_de_barras, numero_da_compra)
146 );
```

Result Grid

Filter Rows:

Export:

Wrap Cell Content:

	Field	Type	Null	Key	Default	Extra
▶	codigo_de_barras	varchar(50)	NO	PRI	NULL	
	numero_da_compra	int	NO	PRI	NULL	
	qtd_de_itens	int	NO		NULL	

1FN) Como podemos ver, todos os atributos da tabela *ItemPedido* são atômicos de modo que cada um pode ser expressado por um único valor.

2FN) Como visto, essa tabela está na forma normal 1 e então cumpre este requisito mínimo para estar na forma normal 2. Sendo as chaves primárias “número da compra” e “código de barras”, e o único atributo complemento das chaves sendo a “quantidade de itens” fica fácil observar que nem o código de barras nem o número da compra definem totalmente funcionalmente a quantidade de itens, pois é necessário diferenciar a compra através do número da compra, e é preciso diferenciar cada produto comprado através de seu código de barra.

3FN) Como visto, essa tabela está na forma normal 2, e então cumpre este requisito mínimo para estar na forma normal 3. Além disso, como existe um único atributo não chave nessa tabela, ela necessariamente está na forma normal 3.

3) Tabela Carrinho


```

122 • CREATE TABLE IF NOT EXISTS Carrinho
123 (
124     numero_da_compra      INT AUTO_INCREMENT PRIMARY KEY NOT NULL,
125     cpf_do_cliente        VARCHAR(30) NOT NULL,
126     data_de_compra        DATE NOT NULL,
127     cpf_do_funcionario    VARCHAR(50) NOT NULL,
128
129     FOREIGN KEY(cpf_do_cliente) REFERENCES Cliente(cpf),
130     FOREIGN KEY(cpf_do_funcionario) REFERENCES Funcionario(cpf)
131
132 );

```

Result Grid

Filter Rows:

Export: 

Wrap Cell Content: ☒

	Field	Type	Null	Key	Default	Extra
▶	numero_da_compra	int	NO	PRI	NULL	auto_increment
	cpf_do_cliente	varchar(30)	NO	MUL	NULL	
	data_de_compra	date	NO		NULL	
	cpf_do_funcionario	varchar(50)	NO	MUL	NULL	

1FN) Como podemos ver, todos os atributos da tabela *Carrinho* são atômicos de modo que cada um pode ser expressado por um único valor.

2FN) Como visto, essa tabela está na forma normal 1 e então cumpre este requisito mínimo para estar na forma normal 2. Além disso, percebemos que todos os atributos do complemento da chave primária (no caso o número da compra), ou seja, o CPF do cliente, a data da compra e o CPF do funcionário são totalmente funcionalmente dependentes da chave.

3FN) Como visto, essa tabela está na forma normal 2, e então cumpre este requisito mínimo para estar na forma normal 3. Além disso, percebemos que todos os atributos não-chave são dependentes não transitivos da chave primária pois:



- Um cliente (portador de um CPF) pode realizar compras em datas diferentes e com funcionários (também portadores de um CPF) diferentes.
- Em uma mesma data, diversas compras podem acontecer, o que implica que apenas a data não é capaz de definir qual cliente realizou uma compra (*Item Pedido*) ou qual funcionário atendeu esse cliente.
- Da mesma forma, um funcionário pode atender diversos clientes e ele pode trabalhar em diversas datas distintas de modo que não é capaz de definir qualquer outro atributo.

Assim, como não há garantias de que um atributo não-chave possa definir outro atributo não-chave, não existe dependência transitiva entre eles. Logo, esta tabela está na terceira forma normal.

4) Tabela Produto

11 • CREATE TABLE IF NOT EXISTS Produto

```
12  (
13      codigo_de_barras    VARCHAR(50) PRIMARY KEY NOT NULL,
14      nome                VARCHAR(50) NOT NULL,
15      valor               FLOAT NOT NULL,
16      marca               VARCHAR(50) NOT NULL,
17      categoria           VARCHAR(50) NOT NULL
18  );
```

<						
Result Grid						
Filter Rows: <input type="text"/>						
Export: 						
Wrap Cell Content: 						
	Field	Type	Null	Key	Default	Extra
▶	codigo_de_barras	varchar(50)	NO	PRI	NULL	
	nome	varchar(50)	NO		NULL	
	valor	float	NO		NULL	
	marca	varchar(50)	NO		NULL	
	categoria	varchar(50)	NO		NULL	

1FN) Como podemos ver, todos os atributos da tabela *Produto* são atômicos de modo que cada um pode ser expressado por um único valor.

2FN) Como visto, essa tabela está na forma normal 1 e então cumpre este requisito mínimo para estar na forma normal 2. Além disso, percebemos que todos os atributos do complemento da chave primária (no caso o código de barras), ou seja, o nome, o valor, a marca e a categoria são totalmente funcionalmente dependentes da chave.

3FN) Como visto, essa tabela está na forma normal 2, e então cumpre este requisito mínimo para estar na forma normal 3. Além disso, percebemos que todos os atributos não-chave são dependentes não transitivos da chave primária pois:

- Um produto, com mesmo nome, pode ter um valor diferente (de acordo com suas especificidades como por exemplo tamanho de uma peça de vestuário), bem como uma marca distinta (como um mouse *gamer* das marcas *TGT* e *RedDragon*), ou mesmo uma categoria diferente (As Crônicas de Nárnia é uma adaptação cinematográfica de um livro com o mesmo nome, por exemplo).
- Claramente, um valor não é suficiente para diferenciar um produto pelo nome visto que podem existir produtos com mesmo preço e nomes distintos. O mesmo se aplica as categorias e marcas.
- Uma marca por sua vez não é capaz de diferenciar o nome de um produto (podem existir diversos produtos da mesma marca), bem como não é capaz de determinar o preço (visto que o mesmo pode variar entre seus produtos) e muito menos a categoria visto que uma marca pode ter diversos produtos na mesma categoria.
- Finalmente, a categoria de um produto não é suficiente para determinar o nome de um produto (dizer que o produto é um livro em nada ajuda a determinar se é *Harry Potter* ou *As Crônicas de Nárnia*), bem como não é capaz de determinar um valor (pois o mesmo é influenciado pelo fabricante, loja, custo de transporte e etc.). A categoria também é incapaz de definir a marca de um produto, afinal, podem existir diversas marcas atuando em um mesmo ramo de produção.

Assim, como não há garantias de que um atributo não-chave possa definir outro atributo não-chave, não existe dependência transitiva entre eles. Logo, esta tabela está na terceira forma normal.

5) Tabela Vestuário

```
13 CREATE TABLE IF NOT EXISTS Vestuario
14 (
15     codigo_de_barras    VARCHAR(50) NOT NULL PRIMARY KEY,
16     tamanho             VARCHAR(50) NOT NULL,
17     cor                 VARCHAR(50) NOT NULL,
18     material            VARCHAR(50) NOT NULL,
19
20     FOREIGN KEY(codigo_de_barras) REFERENCES Produto(codigo_de_barras)
21 );
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

Field	Type	Null	Key	Default	Extra
codigo_de_barras	varchar(50)	NO	PRI	NULL	
tamanho	varchar(50)	NO		NULL	
cor	varchar(50)	NO		NULL	
material	varchar(50)	NO		NULL	

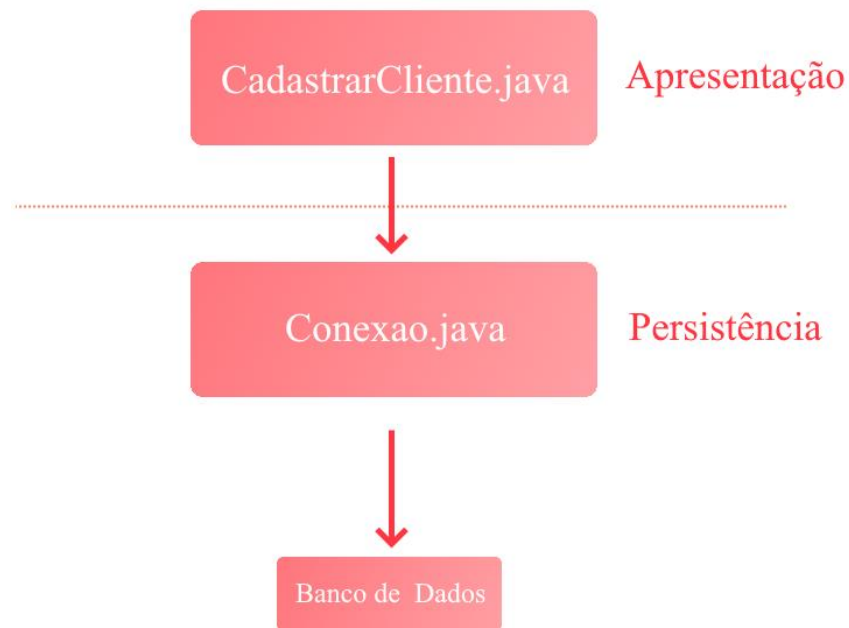
1FN) Como podemos ver, todos os atributos da tabela *Vestuário* são atômicos de modo que cada um pode ser expressado por um único valor.

2FN) Como visto, essa tabela está na forma normal 1 e então cumpre este requisito mínimo para estar na forma normal 2. Além disso, percebemos que todos os atributos do complemento da chave primária (no caso o código de barras), ou seja, o nome, o valor, a marca e a categoria são totalmente funcionalmente dependentes da chave.

3FN) Como visto, essa tabela está na forma normal 2, e então cumpre este requisito mínimo para estar na forma normal 3. Além disso, percebemos que todos os atributos não-chave são dependentes não transitivos da chave primária pois:

- O tamanho da peça não é o suficiente para definir a peça em si, pois podem existir dois produtos do mesmo tamanho, mas com cores e matérias diferentes. Por exemplo, com cores e materiais diferentes, como por exemplo pode existir simultaneamente um vestido vermelho de cetim e um vestido verde de renda, ambos tamanho G.
- A cor também não define o produto unicamente, pois podem existir duas peças com a mesma cor de tamanhos diferentes, assim como material. Por exemplo, pode existir um sapato 42 e preto de couro, junto com uma bota 42 e marrom de borracha.
- Por último o material também não faz parte de uma possível transitividade, pois podem existir dois produtos de mesmo material e cores e tamanhos diferentes. Podem existir simultaneamente no banco uma calça Jeans azul escuro de tamanho PP, e uma calça Jeans azul claro M, por exemplo.

Diagrama da Camada de Mapeamento para a tabela *Cliente*



Aqui podemos perceber como se dá o processo de iteração entre o banco de dados e a interface gráfica. Todas as informações são resgatadas na camada de apresentação e encaminhadas para a camada de persistência. Esta por sua vez realiza a conexão com o banco de dados que efetivamente processa os dados.