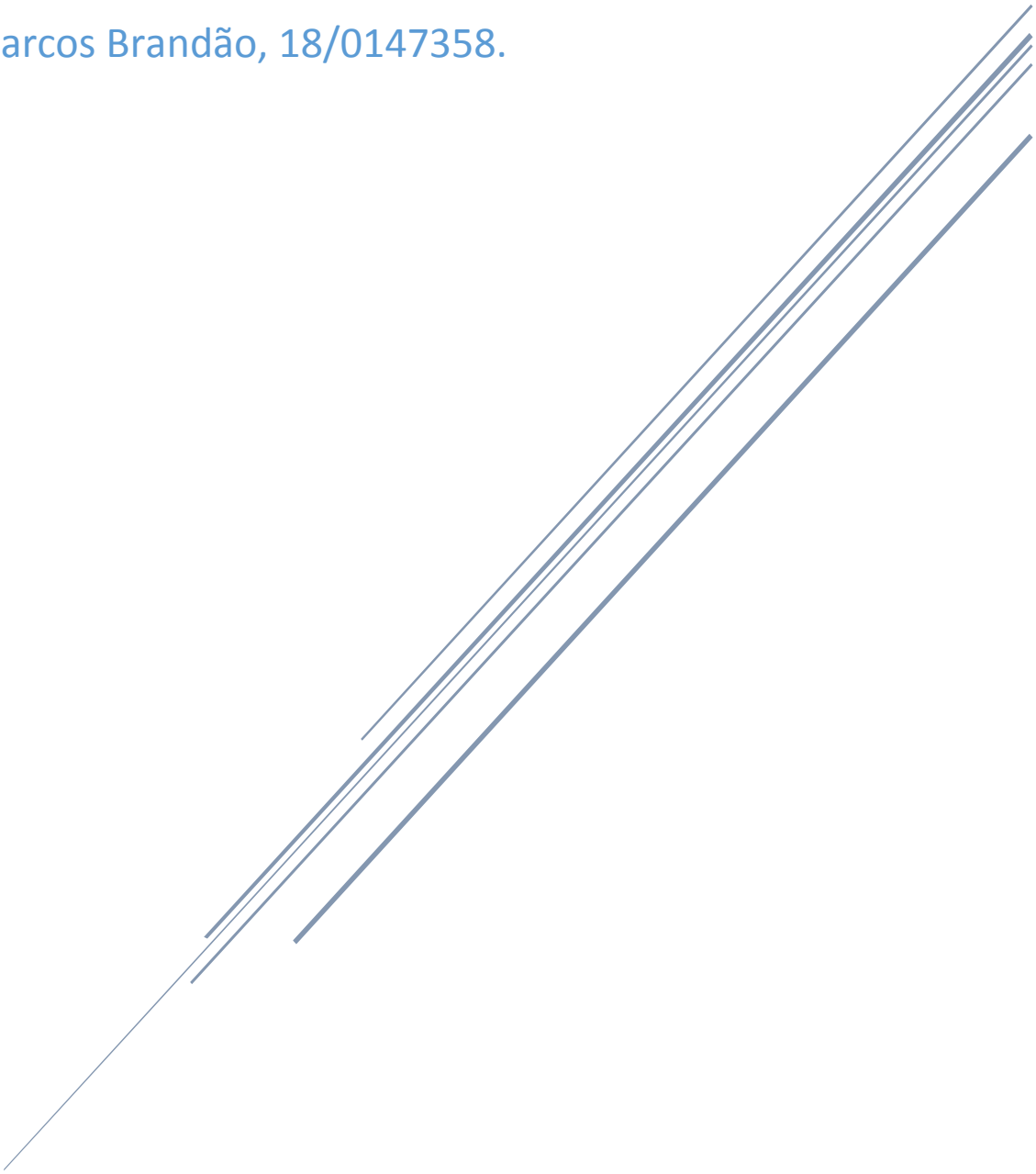
PROJETO FINAL DE BANCO DE DADOS

Lucas da Silva Souza, 16/0013020;

Gustavo Costa de Souza, 15/0128576;

Victor Alves de Carvalho, 16/0147140;

Italo Marcos Brandão, 18/0147358.



Universidade de Brasília.
Banco de Dados, Turma B.

Sumário

1. Introdução

A companhia VILG S.A. é uma empresa logística cuja função é transportar cargas dentro do território nacional; nossa sede se localiza em Brasília, Distrito Federal. Fazemos entregas para localidades pedidas pelo cliente – seja Pessoa Física (CPF) ou Pessoa Jurídica (CNPJ) - através de nossos motoristas e veículos de cargas.

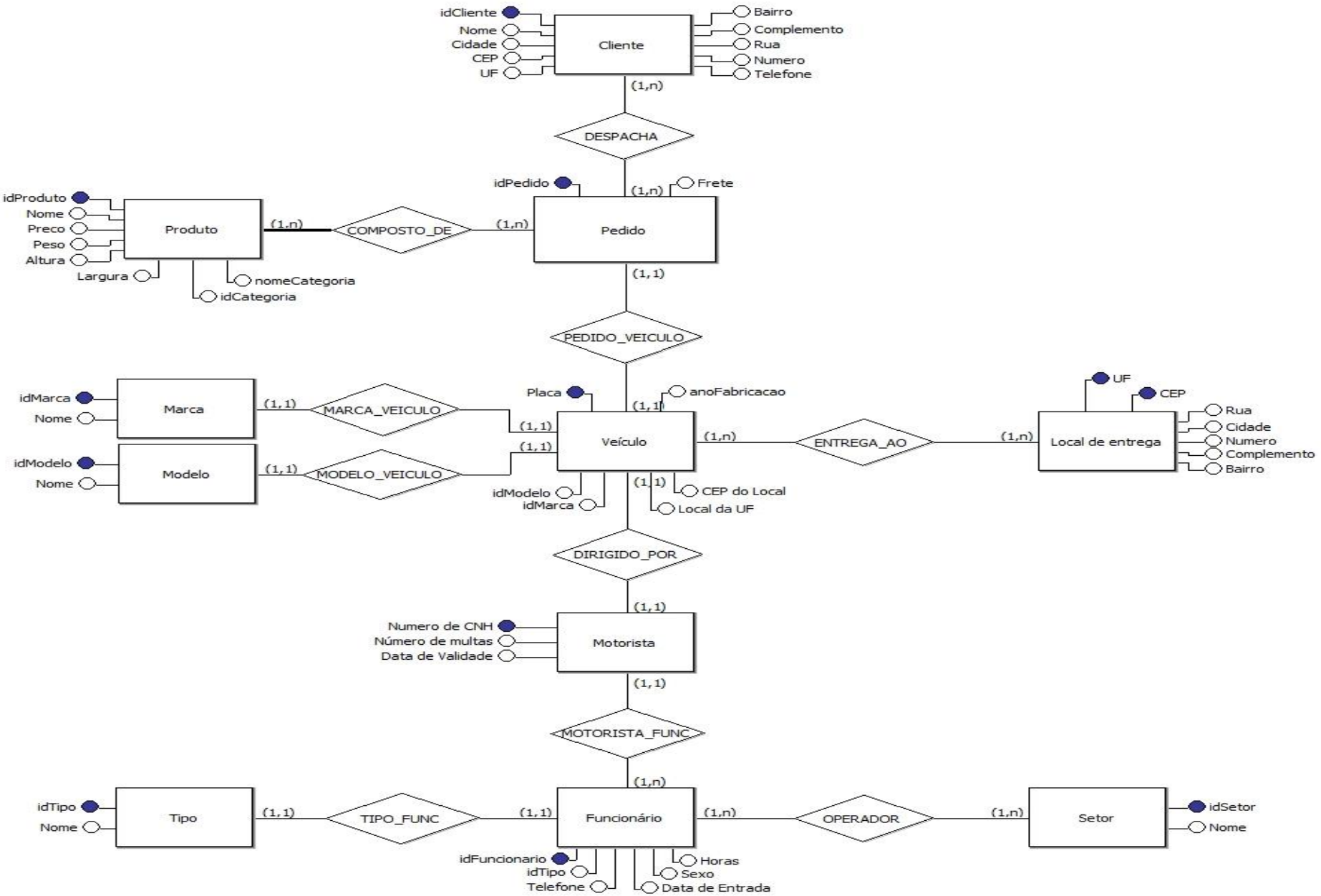
Nossa empresa possui um sistema logístico baseado em:

1.1. Cadastro do cliente na base de dados – seja Pessoa Física ou Pessoa Jurídica – para que seja anotado na nota do pacote;
1.2. Recebimento do pacote, por cliente (Pessoa Física ou Pessoa Jurídica). É identificado as características do pacote, tais como a categoria (frágil, eletrônico, documento, inflamável), suas dimensões, seu peso, local de entrega e o valor do frete;
1.3. Destinação do pacote – caso seja pago - para um veículo de carga junto com outras encomendas que estarão na rota do motorista, tarefa do operador de carga;
1.4. Efetuação do transporte da encomenda, feita por um motorista, que pode dirigir em trecho rodoviário ou urbano;
1.5. Entrega ao destinatário seguindo as informações prestadas pelo cliente.

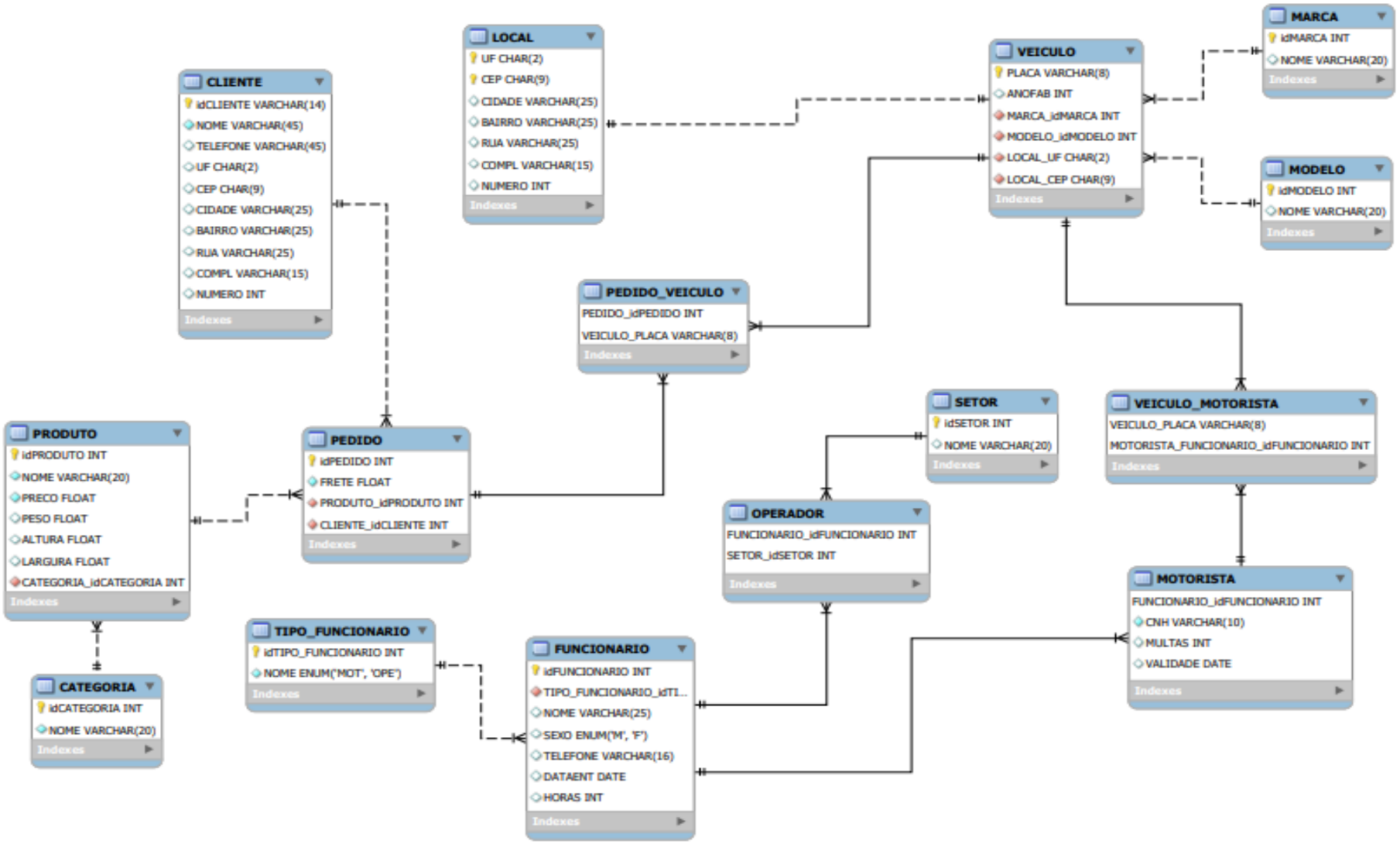
Este projeto tem como intuito criar uma base de dados relacional para registro de clientes, motoristas, operadores, produtos, veículos e locais de entrega; todos estes em conjunto, garantindo a eficiência da operação e formalizando o processo logístico de fretes, considerando as condições internas da companhia e fatores externos, como a tabela mínima de frete – imposta pela Agência Nacional de Transportes Terrestres -, o respeito ao Peso Bruto Total do implemento urbano e rodoviário e as condições das rodovias para que a encomenda chegue intacta ao destinatário no prazo determinado.

O projeto estará funcionando em caráter funcional, com a projeção de realizar futuras modificações na base de dados, caso seja pedido pelos stakeholders. Nosso código-fonte está disponível na plataforma GitHub, cujo link está anexado.

2. Diagrama Entidade Relacionamento



3. Modelo Relacional



4. Cinco consultas em álgebra relacional, onde cada consulta envolva pelo menos 3 tabelas.
    1. Funcionários que são motoristas de caminhão 'Scania'
    Tb_Temporaria1 ← FUNCIONARIO x MOTORISTA x VEICULO_MOTORISTA x VEICULO x MARCA;
    π FUNCIONARIO.nome (σ MARCA.nome = ‘Scania’
    AND
    VEICULO.placa = VEICULO_MOTORISTA.veiculo_placa AND
    MOTORISTA.funcionario_id = VEICULO_MOTORISTA.motorista_funcionario_id)
    (Tb_Temporaria1);

    2. Quais os nomes dos clientes que pediram eletrônicos?
    Tb_Temporaria2 ←CLIENTE x PEDIDO x PRODUTO x CATEGORIA;
    Tb_Selecao2 ← σ CATEGORIA.nome='Eletronicos' (Tb_Temporaria2);
    Tb_Definitiva2 ← π CLIENTE.nome, PEDIDO.idProduto, PRODUTO.nome, CATEGORIA.nome (Tb_Selecao2);

    3. Relação de pedidos enviado a cidade de Goianápolis, com marca e modelo do veículo utilizado;
    Tb_Temporaria3 ← PEDIDO x PRODUTO x PEDIDO_VEICULO x VEICULO x MARCA x MODELO x LOCAL;
    Tb_Selecao3 ← σ LOCAL.cidade = ‘Goianapolis’ (Tb_Temporaria3);
    Tb_Definitiva3 ← π PEDIDO.idPedido, PRODUTO.nome, LOCAL.cidade, VEICULO.placa, MARCA.nome, MODELO.nome;

    4. Liste todos os pedidos com destino ao DF.
    Tb_Temporaria4 ← PEDIDO x PEDIDO_VEICULO x VEICULO;
    Tb_Selecao4 ← σ VEICULO.local_uf = ‘DF’
    AND
    PEDIDO_VEICULO.pedido_idpedido = PEDIDO.idpedido
    AND
    PEDIDO_VEICULO.veiculo_placa = VEICULO.placa
    (Tb_Temporaria4);
    Tb_Definitiva4 ← πPEDIDO.idpedido, PRODUTO.nome (Tb_Selecao4);

    5. Relação de clientes cujo pedido possui o peso do produto acima de 15 kg
    Tb_Temporaria5 ← CLIENTE x PEDIDO x PRODUTO x CATEGORIA;
    Tb_Selecao5 ← σPRODUTO.peso=15000 (Tb_Temporaria5);
    Tb_Definitiva5 ← π CLIENTE.nome, PEDIDO.idPedido, CATEGORIA.nome, PEDIDO.peso;

5. Avaliação das formas normais em cinco tabelas

O processo de Normalização, baseado em Formas Normais, tem como intuito:

a. Na Primeira Forma Normal, analisar atributos possivelmente divisíveis, defini-los como atômicos (valor único) e definir chaves primárias;
b. Na Segunda Forma Normal, definir dependências de atributos para cada chave primária;
c. Na Terceira Forma Normal, eliminar redundâncias.

Para isto, nosso código-fonte se encontra, atualmente, na Terceira Forma Normal, com a eliminação de redundâncias e trafegabilidade efetivas de dados entre as tabelas. Também houve eliminação de alguns atributos propostos pelo Diagrama Entidade Relacionamento (DER), por haver a possibilidade do Sistema de Gestão de Banco de Dados gerar automaticamente a data e o horário do lançamento, através de timestamp.

Como exemplo, escolhemos as entidades referentes ao cliente, as características do pacote e o local de entrega.

1) Primeiro caso: tabela **CLIENTE**
Atributo chave: idCLIENTE.
Atributos: NOME, TELEFONE, UF, CEP, CIDADE, BAIRRO, RUA, COMPL, NUMERO.

Primeira Forma Normal: Não há atributos divisíveis, somente atributos atômicos.

Segunda Forma Normal: As dependências estão de acordo com a chave primária proposta nesta tabela.
{idCliente} ⟶ {NOME, TELEFONE, UF, CEP, CIDADE, BAIRRO, RUA, COMPL, NUMERO}.

Terceira Forma Normal: Não há redundâncias da chave primária com os atributos compostos na tabela.

Portanto, está normalizado na Primeira, Segunda e Terceira Forma Normal.

2) Segundo caso: tabela **PEDIDO**
Atributo chave: idPEDIDO.
Atributos: FRETE, PRODUTO_idPRODUTO <FK> e CLIENTE_idCLIENTE <FK>.

Primeira Forma Normal: Não possui atributos divisíveis nesta tabela, portanto, está normalizado.

Segunda Forma Normal: Todos os atributos compostos nesta tabela estão de acordo com a proposta de dependência da chave primária.
{idPEDIDO} ⟶ {FRETE, PRODUTO_idPRODUTO <FK>, CLIENTE_idCLIENTE <FK>}.

Terceira Forma Normal: Não há redundâncias nos atributos desta tabela.

Portanto, esta tabela se encontra normalizada.

3) Terceiro caso: tabela **PRODUTO**
Atributo chave: idPRODUTO.

Atributos: NOME, PRECO, PESO, ALTURA, LARGURA, CATEGORIA_idCATEGORIA <FK>.

Primeira Forma Normal: Não há atributos que possam ser divisíveis, todos estão atômicos.

Segunda Forma Normal: Todos os atributos presentes nesta tabela são dependentes da chave primária.
{idPRODUTO} ⟶ {NOME, PRECO, PESO, ALTURA, LARGURA, CATEGORIA_idCATEGORIA <FK>}.

Terceira Forma Normal: A única redundância que poderia ocorrer está na classificação da categoria do produto, que foi corrigida previamente com base no nosso Diagrama Entidade Relacionamento (DER). A mesma está referenciando à tabela **CATEGORIA**, que trataremos no item seguinte. No mais, está normalizado na regra da Terceira Forma Normal.

Portanto, esta tabela está normalizada.

4) Quarto caso: tabela **CATEGORIA**
Atributo chave: idCATEGORIA.
Atributo: NOME

Primeira Forma Normal: Esta tabela está normalizada, pois o ID da categoria do produto é único e auto incrementável, ou seja, com valor atômico.

Segunda Forma Normal: A dependência desta tabela se resume apenas ao nome da categoria, portanto, está normalizado nesta questão.
{idCATEGORIA} ⟶ {NOME}.

Terceira Forma Normal: Não é possível haver redundâncias nesta tabela, portanto, também está normalizado.

Esta tabela satisfaz as três condições de normalização.

5) Quinto caso: tabela **LOCAL**
Atributos chave: UF, CEP.
Atributos: CIDADE, BAIRRO, RUA, COMPL, NUMERO.

Primeira Forma Normal: Não há divisibilidade dos atributos compostos com as chaves primárias, portanto, os atributos são atômicos.

Segunda Forma Normal: Todos os atributos compostos nesta tabela são dependentes das chaves primárias.
{UF, CEP} ⟶ {CIDADE, BAIRRO, RUA, COMPL, NUMERO}.

Terceira Forma Normal: Não há divisibilidade transitiva dos atributos desta tabela.

Como conclusão, esta tabela satisfaz as três condições de normalização.

6. Diagrama da camada de mapeamento para uma tabela do banco de dados

O Diagrama Model View Controller (MVC), ou Controlador do Modelo de Visão, é um padrão de projeto, baseado no desenvolvimento de interfaces - que são divididos em Modelo, Controlador e Visão - permitindo a separação de representações das informações. O diagrama auxilia no desenvolvimento da interface gráfica em linguagens Web, como JavaScript, C++, Python e HTML.

Os alicerces destes diagramas são compostos de:

- **Modelo:** contém todos os dados de um aplicativo e tem a capacidade de manipular esses dados. Não tem interação com a Visão, mas se comunica com o Controlador;
- **Controlador:** Recebe solicitações do navegador, entra em contato com o Modelo para qualquer dado que possa precisar e, em seguida, captura a Visualização adequada para exibir esses dados para o usuário;
- **Visão:** tudo o usuário vê no navegador. Geralmente em HTML, CSS e Javascript.

O funcionamento deste diagrama mostra como é feita a comunicação entre o usuário e a base de dados do sistema. Confira:

1) O usuário realiza uma pedido de consulta;
2) O controlador realiza o contato com o modelo;
3) O modelo, por sua vez, coleta as informações pedidas dentro da base de dados;
4) Quando coletado, retorna a resposta ao controlador;
5) O controlador, por sua vez, entrega o pedido para o modo de Visão;
6) Por fim, a Visão fica por conta de mostrar o dado para o usuário.

No contexto deste projeto, abordaremos o mapeamento sobre a tabela **PRODUTO**, feita sobre a linguagem de programação Java. Estaremos destacando abaixo as extensões Java utilizadas no trabalho, tendo em vista as definições acima sobre Controle, Modelo e Visão:

- **Controlador:** Produto.java;
- **Modelo:** ProdutoDAO.java;
- **Visão:** TelaCadastroProduto.java e TelaRemoverProduto.java.

Utilizamos a IDE Netbeans v8.0.2, para que a implementação em Java pudesse ser efetuada. Também adotamos o servidor local WAMP para simular o funcionamento do banco de dados, simulando um serviço real em servidores empresariais.