



Universidade do Minho
Mestrado Integrado em Engenharia Informática
Licenciatura em Ciências da Computação

Unidade Curricular de Bases de Dados

Ano Letivo de 2018/2019

DistribuMinho

**Diogo Braga A82547, João Silva A82005,
Ricardo Caçador A81064, Ricardo Milhazes A81919**

Novembro, 2018

BD

Data de	
Recepção	
Responsável	
Avaliação	
Observações	

DistribuMinho

**Diogo Braga A82547, João Silva A82005,
Ricardo Caçador A81064 , Ricardo Milhazes A81919**

Novembro, 2018

O grupo de trabalho dedica este projeto a todos os seus familiares, amigos e amantes, que nos momentos de maior necessidade estiveram lá para amparar toda a tormenta causada pelo enorme desgaste, proveniente do esforço continuado da execução deste projeto.

Por último, o grupo deixa o profundo agradecimento à empresa real na qual se baseou a ideia de todo este projeto, *Soutivinhos*.

Resumo

Este relatório apresenta todo o processo efetuado para a realização de um sistema de base de dados referente a uma distribuidora de alimentos e bebidas, denominada de DistribuMinho.

Inicialmente demonstramos o caso de estudo e a contextualização do mesmo perante as necessidades no nosso quotidiano. Desta forma, são também apresentados vários requisitos a cumprir para um bom funcionamento do Sistema Geral de Base de Dados.

Tendo em conta os requisitos, de seguida é apresentado o modelo conceptual analisando também vários aspectos como a representação das entidades, relacionamentos, atributos, etc.

Construímos também o modelo lógico, e apresentamos na mesma secção tópicos como a validação do modelo através da normalização, através das interrogações do utilizador, através das transações estabelecidas, etc. Desta forma é garantida a realização de um eficiente SGBD.

O último modelo realizado é o físico, e nesta secção são também apresentados alguns exemplos de queries desenvolvidas, que derivam em *procedures*, *transactions*, *views*, etc.

Por último, concluímos o projeto com uma apreciação crítica dos diferentes itens abordados e uma previsão para o futuro da nossa base de dados.

Área de Aplicação: Sistemas de Bases de Dados

Palavras-Chave: Bases de Dados Relacionais, MySQL, Modelo Conceptual, Modelo Lógico, Modelo Físico.

Índice

Resumo	i
1. Introdução	8
1.1. Contextualização	8
1.2. Apresentação do Caso de Estudo	8
1.3. Motivação e Objetivos	9
1.4. Análise de Viabilidade do Projeto	10
1.5. Estrutura do Relatório	10
2. Levantamento de Requisitos	11
2.1. Método de levantamento e de análise de requisitos adotado	11
2.2. Requisitos Levantados	11
2.2.1 Requisitos de Descrição	11
2.2.2 Requisitos de Exploração	12
2.2.3 Requisitos de Controlo	12
2.3. Perfis de Utilização	13
2.4. Análise Geral dos Requisitos	13
3. Modelação Conceptual	15
3.1. Apresentação da abordagem de modelação realizada	15
3.2. Identificação e caracterização das entidades	15
3.3. Identificação e caracterização dos relacionamentos	16
3.4. Identificação e associações dos Atributos com as Entidades e Relacionamentos	17
3.4.1 Armazém	17
3.4.2 Produto	18
3.4.3 Cliente	18
3.4.4 Encomenda	18
3.4.5 Veículo Transportador	18
3.4.6 Relacionamento Armazém-Produto	19
3.4.7 Relacionamento Produto-Encomenda	19
3.5. Domínio dos Atributos	19

<i>3.6. Definição das Chaves Candidatas, Primárias e Alternativas</i>	23
<i>3.7. Verificação das Redundâncias no Modelo</i>	24
3.7.1 Reexaminar Relacionamentos (1:1)	24
3.7.2 Remover Relacionamentos Redundantes	24
3.7.3 Considerar Dimensão Temporal	24
<i>3.8. Apresentação e explicação do diagrama ER</i>	25
<i>3.9. Validação do Modelo Conceptual contra Transações do Utilizador</i>	26
3.9.1 Adicionar um novo Cliente	26
3.9.2 Adicionar um novo Produto	26
3.9.3 Adicionar um veículo transportador novo	26
3.9.4 Atualizar informações de um veículo transportador	26
3.9.5 Adicionar um novo produto	26
3.9.6 Adicionar uma nova encomenda	26
3.9.7 Atualizar a data de entrega da encomenda	27
<i>3.10. Validação do Modelo de Dados com o Utilizador</i>	27
4. Modelação Lógica	28
<i>4.1. Construção e validação do modelo de dados lógico</i>	28
<i>4.2. Desenho do modelo lógico</i>	30
<i>4.3. Validação do modelo através da normalização</i>	30
4.3.1 Unnormalized Form (UNF)	33
4.3.2 First Normal Form (1NF)	33
4.3.3 Second Normal Form (2NF)	33
4.3.4 Third Normal Form (3NF)	34
<i>4.4. Validação do modelo com interrogações do utilizador</i>	34
4.4.1 Permitir a consulta das informações de um Produto.	34
4.4.2 Permitir a consulta das informações dum Cliente.	34
4.4.3 Permitir a consulta de informações duma Encomenda.	34
4.4.4 Permitir a consulta de informações de um Armazém.	35
4.4.5 Permitir a consulta das informações do Veículo Transportador.	35
4.4.6 Listar as Encomendas designadas a um cliente em específico.	35
4.4.7 Listar as Encomendas que decorrem num intervalo de datas.	35
4.4.8 Listar as Encomendas que decorreram numa dada cidade.	35
4.4.9 Listar as Encomendas que foram transportadas por um Veículo Transportador.	36
4.4.10 Listar as Encomendas que têm um determinado Produto.	36
4.4.11 Informar sobre o balanço crédito numa determinada data.	36

4.4.12 Filtrar os Veículos Transportadores que necessitam de ser substituídos, no mês corrente.	36
4.4.13 Filtrar os Veículos Transportadores que necessitam de fazer a inspeção, no mês corrente.	37
4.4.14 Filtrar os Veículos Transportadores que necessitam de renovar o imposto de selo, no mês corrente.	37
4.4.15 Filtrar os Veículos Transportadores que necessitam de renovar o seguro, no mês corrente.	37
4.5. Validação do modelo com as transações estabelecidas	37
4.5.1 Adicionar um novo Veículo Transportador	37
4.5.2 Adicionar um novo Cliente	38
4.5.3 Adicionar uma nova Encomenda	38
4.5.4 Adicionar um novo Produto	38
4.5.5 Adicionar um novo Tipo de Produto	38
4.5.6 Adicionar um novo Tipo de Cliente	38
4.5.7 Adicionar um novo Contacto	39
4.5.8 Indicar que um Produto está presente numa Encomenda	39
4.5.9 Indicar que um Produto está recolhido num Armazém	39
4.6. Verificação da integridade das restrições	39
4.6.1 Dados Necessários	40
4.6.2 Domínio das Restrições dos Atributos	40
4.6.3 Cardinalidade	40
4.6.4 Integridade das Entidades	41
4.6.5 Integridade Referencial	41
4.6.6 Restrições Gerais	41
4.7. Revisão do modelo lógico com o utilizador	42
4.8. Verificar futuro crescimento	42
5. Implementação Física	44
5.1. Seleção do sistema de gestão de bases de dados	44
5.2. Tradução do esquema lógico para o sistema de gestão de bases de dados escolhido em SQL	44
5.3. Tradução das interrogações do utilizador para SQL	50
5.3.1 Permitir a consulta das informações dum Cliente	50
5.3.2 Listar as Encomendas que decorreram num intervalo de datas	50
5.3.3 Listar os produtos que são dirigidos para um Cliente	50

5.3.4 Informar sobre os créditos numa determinada data	51
<i>5.4. Tradução das transações estabelecidas para SQL</i>	51
5.4.1 Permitir ao funcionário de escritório adicionar um novo cliente	51
5.4.2 Permitir ao motorista adicionar data de entrega e a distância da encomenda	52
<i>5.5. Escolha, definição e caracterização de índices em SQL</i>	52
<i>5.6. Estimativa do espaço em disco da base de dados e taxa de crescimento anual</i>	53
5.6.1 Taxa de crescimento anual	56
<i>5.7. Definição e caracterização das vistas de utilização em SQL</i>	56
5.7.1 Filtrar os Veículos Transportadores que necessitam de ser substituídos	56
5.7.2 Filtrar os Veículos Transportadores que necessitam de renovar o imposto de selo	57
5.7.3 Listar Encomendas pendentes	57
<i>5.8. Definição e caracterização dos mecanismos de segurança em SQL</i>	58
<i>5.9. Revisão do sistema implementado com o utilizador</i>	59
6. Conclusões e Trabalho Futuro	60
7. Sistema de Base de Dados Não Relacional	62
<i>7.1. Justificação da utilização de um sistema NoSQL</i>	62
7.1.1 Armazenamento de Dados	62
7.1.2 Estrutura e Flexibilidade	63
7.1.3 Escalabilidade	63
7.1.4 Propriedades ACID	63
7.1.5 Vantagens	63
<i>7.2. Identificação e descrição dos objetivos da base de dados, em termos de aplicações e de utilizadores</i>	64
<i>7.3. Questões que serão realizadas sobre o SBD NoSQL</i>	64
<i>7.4. Esquema de Base de Dados</i>	65
<i>7.5. Objetos de Dados para Alimentar SBD NoSQL</i>	66
<i>7.6. Processo de Migração</i>	68
7.6.1 Extração de Dados	68
7.6.2 Transformação de Dados	69
7.6.3 Carregamento de Dados	69
<i>7.7. Detalhes da Extração dos Dados</i>	70
7.7.1 Criação dos Nós	70

7.7.2 Criação dos Relacionamentos entre Nós	72
<i>7.8. Queries em Neo4j</i>	74
7.8.1 Permitir a consulta das informações dum Cliente	74
7.8.2 Listar as Encomendas que decorreram num intervalo de datas	74
7.8.3 Listar os produtos que são dirigidos para um Cliente	74
<i>7.9. Conclusão</i>	75
Referências	76
Lista de Siglas e Acrónimos	77

Índice de Figuras

Figura 1 - Modelo Conceptual	25
Figura 2 - Modelo Lógico	30
Figura 3 - Create Schema DistribuMinho	44
Figura 4 - Create Table Encomenda	45
Figura 5 - Create Table Produto	45
Figura 6 - Create Table Tipo_Produto	46
Figura 7 - Create Table Tipo_Cliente	46
Figura 8 - Create Table Armazem	46
Figura 9 - Create Table Contacto	47
Figura 10 - Create Table Cliente	47
Figura 11 - Create Table Encomenda	48
Figura 12 - Create Table VeiculoTransportador	48
Figura 13 - Create Table Armazem_Produto	49
Figura 14 - Create Table Produto_Encomenda	49
Figura 15 - Procedure InformacoesDeUmCliente	50
Figura 16 - Procedure EncomendasNumIntervaloDatas	50
Figura 17 - Procedure ProdutosDasEncomendasParaUmCliente	50
Figura 18 - Procedure CreditosNumaData	51
Figura 19 - Transaction AdicionarNovoCliente	51
Figura 20 - Transaction AdicionarDataEntrega	52
Figura 21 - Vista VeiculosNecessitamSubstituicao	57
Figura 22 - Vista VeiculosRenovarImpostoSelo	57
Figura 23 - Vista EncomendasPendentes	57
Figura 24 - Criação de Utilizadores	58
Figura 25 - Permissão para um Funcionário de Escritório	58
Figura 26 - Permissão para um Gestor	58
Figura 27 - Médo de extração de dados dos Clientes	69
Figura 28 - Classe Cliente	69
Figura 29 - Método de Carregamento de Clientes	70
Figura 30 - Código de Criação de Nós Cliente	70
Figura 31 - Código de Criação de Nós do Tipo Produto	71
Figura 32 - Código de Criação de Nós do Tipo Encomenda	71
Figura 33 - Código de Criação de nós do Tipo Armazém	71
Figura 34 - Código de Criação de Nós do Tipo Veículo Transportador	72
Figura 35 - Criação de Relacionamentos :PERTENCE_A	72
Figura 36 - Criação de Relacionamentos :ARMAZENADO_EM	72
Figura 37 - Criação de Relacionamentos :INCLUIDO_EM	73
Figura 38 - Criação de Relacionamentos :DESIGNADA_A	73

Figura 39 - Criação de Relacionamentos :TRANSPORTA	73
Figura 40 - Querie Neo4j	74
Figura 40 - Querie Neo4j	74
Figura 41 - Querie Neo4j	74
Figura 41 - Querie Neo4j	74
Figura 42 - Querie Neo4j	74
Figura 42 - Querie Neo4j	74

Índice de Tabelas

Tabela 1 - Domínio dos atributos das entidades	22
Tabela 2 - Domínio dos atributos dos relacionamentos	23
Tabela 3 - Relações do Modelo Lógico	30

1. Introdução

Nesta secção será feita a apresentação, algo superficial, do tema escolhido, bem como os motivos que nos levaram a escolher tal tema. Será também realizada uma análise à viabilidade do projeto, e por último, mostramos, de forma muito breve, a estrutura de desenvolvimento do relatório, com o objetivo de cada secção.

1.1. Contextualização

No nosso dia-a-dia, é usual depararmo-nos, em estradas (ou não), com várias transportadoras, que fazem distribuição dos mais variados artigos, desde produtos alimentares, animais vivos, móveis, materiais perigosos, etc. Fica assim subentendido que qualquer transportadora de qualquer tipo pode fazer 10, 50, 100, 1000 transportes de mercadoria por dia, pelo que fica complicado gerir grandes quantidades de dados manualmente e possivelmente com ferramentas de baixa qualidade.

Dado este problema, ganha ênfase a necessidade de organizar os dados relativos a cada transporte, de modo a que os gestores de cada transportadora possam saber por exemplo, o que foi transportado para um determinado destino, ou quanto se ganhou num determinado espaço de tempo, etc.

1.2. Apresentação do Caso de Estudo

O grupo decidiu então implementar uma base de dados para uma empresa de distribuição de bebidas e produtos alimentares, cujo nome é DistribuMinho, de forma a que esta possa gerir e consultar distribuições a seu bel-prazer.

A DistribuMinho possui vários armazéns. Os armazéns possuem um número de identificação, um número de camiões e uma morada. Nestes são armazenados os produtos provenientes dos fornecedores. Os produtos têm um nome, um número de identificação, um tipo e um preço. São encomendados pelos funcionários de escritório a pedido do chefe do armazém quando este repara na necessidade de obtenção devido à escassez em armazém.

O armazém é organizado pelos funcionários de armazém e também pelos motoristas da empresa.

A empresa recebe pedidos para fornecimentos provenientes de clientes. Estes clientes são caracterizados por um nome, um NIF, uma morada (composta por país, cidade e rua), um endereço eletrónico e um tipo, que descreve se são do tipo minimercado, ou supermercado, ou bar, ou restaurante, etc.

A empresa satisfaz os pedidos de fornecimentos dos clientes quando lhes entrega o que foi pedido, essa entrega é feita por um processo que aqui chamaremos de encomenda e consequente essa encomenda é feita por um veículo de transporte de mercadorias. Uma encomenda é caracterizada então pelo número de identificação, uma distância e uma data.

Por último, o veículo que realiza o transporte é caracterizado pela matrícula, pelo número de identificação, o número de quilómetros realizados, pelo ano de fabrico, pela marca, pela classe e pelo número de quilómetros total.

Com isto é pretendido que a Base de Dados a construir guarde todo o tipo de dados que a empresa necessite, para de uma forma mais simplista conseguir aceder a todo o seu negócio.

1.3. Motivação e Objetivos

Assistindo ao crescimento no mercado da DistribuMinho, e com o sucessivo aumento de clientes, foi necessário que a empresa alargasse ao número de funcionários e veículos para realizar todas as distribuições. O maior problema surgiu perante os funcionários de escritório quando se depararam com a enorme dificuldade que tinham em organizar todo o seu negócio.

Devido aos equipamentos rudimentares que a empresa utilizava, tornou-se bastante difícil para os funcionários de escritório organizar os dados de forma perceptível, consequentemente os gestores viram a sua vida muito dificultada pois, era bastante difícil tomar decisões perante a má organização dos dados. Na parte do armazém, os chefes não conseguiam controlar todo o fluxo de produtos que tanto entravam no armazém como na hora seguinte saíam.

O crescimento empresarial da DistribuMinho levou a que escolhessem mecanismos mais profissionais de tratamento e consulta de dados, pelo que a solução foi implementar um sistema de base de dados com o objetivo de satisfazer todos os departamentos da empresa.

Uma vez que um dos objetivos deste sistema de base de dados é ser utilizado como auxílio de uma aplicação *mobile*, torna possível que sejam os camionistas a finalizarem o processo de distribuição, pelo que bastante carga de trabalho é retirada dos funcionários de escritório.

O sistema de Base de Dados torna o processo de análise de dados por parte dos gestores facilitado, pelo que os dados que eles consultam têm garantia de correção.

1.4. Análise de Viabilidade do Projeto

Lendo atentamente os pontos anteriores, percebe-se que a decisão de implementação da base de dados na empresa não é direcionada ao impulsionamento do negócio no momento, mas sim uma medida que vai ajudar ao crescimento sustentado da empresa a longo prazo.

Inicialmente a empresa terá que acarretar com vários custos, começando pelo custo de implementação e manutenção, tanto do sistema de base de dados como da aplicação *mobile* e da aplicação *desktop*.

De maneira a colmatar os custos associados espera-se que os gestores, baseando-se na qualidade de dados oferecida, consigam tomar as melhores decisões de estruturação da empresa, de forma a que se torne rentável a decisão de implementação deste sistema.

1.5. Estrutura do Relatório

Como foi depreendido da leitura da presente secção, esta introduz o problema a resolver e consequentemente o projeto a realizar. Na secção 2 serão levantados todos os requisitos necessários para a realização de um sistema de base de dados que melhor retrate o problema a resolver.

Na secção 3 iniciar-se-á a modelação do SGBD com o desenho do modelo conceptual, onde se identificam as entidades, atributos e relacionamentos. Na secção 4, derivar-se-á o modelo conceptual para o modelo lógico, normalizando-o, e garantiremos que é um bom modelo para sustentar o modelo físico. Na secção 5 passaremos então, do modelo lógico para o físico, realizando alguns *Procedures*, *Views*, *Triggers* e *Transactions*.

Por último na secção 6 serão apresentadas as reflexões finais e uma apreciação do sistema construído.

2. Levantamento de Requisitos

Nesta secção serão apresentados os requisitos que foram levantados e validados com os clientes bem como a forma como foram adquiridos. Por último será feita uma análise alargada a estes requisitos.

2.1. Método de levantamento e de análise de requisitos adotado

Tendo em conta os aspetos que foram realçados na secção de Motivação e Objetivos (1.3), fizemos o levantamento de requisitos de forma a facilitar o funcionamento da DistribuMinho. Assim, a velocidade e eficácia das tarefas da empresa (transporte e carregamento de camiões) irá ser, exponencialmente, melhorada.

Este processo teve em base duas técnicas de levantamento de requisitos, sendo estas *BrainStorming* e um levantamento orientado a pontos de vista, de forma a inserir todos os tipos de ações necessárias.

2.2. Requisitos Levantados

De seguida, enumeram-se os requisitos que o sistema de base de dados deverá suportar, tanto a nível de descrição, como de exploração e de controlo.

2.2.1 Requisitos de Descrição

1. Um Armazém tem um número de armazém, uma morada e um número de camiões.
2. Um Armazém tem vários stocks de produtos específicos.
3. Um Produto é definido por um número de identificação, um nome, um tipo e um preço.
4. Um tipo de produto pode ser charcutaria, congelados, bebidas, etc.
5. Um Cliente possui um nome, um NIF, uma morada, um endereço eletrónico, um tipo e, possivelmente, vários contactos.

6. Um tipo de cliente pode ser definido como Minimercado, Supermercado, Bar, Restaurante, etc.
7. Uma Encomenda é caracterizada por um número de identificação, uma distância, uma data de pedido e uma data de entrega.
8. Uma Encomenda leva uma quantidade de vários produtos.
9. Um Veículo Transportador tem uma matrícula, o número de quilómetros realizados, um ano de fabrico, uma marca, um condutor, o número de quilómetros máximo por encomenda, a data de renovação do seguro, a data da inspeção, a data de renovação do imposto de selo e um tipo.
10. Uma morada é composta por uma cidade e uma rua.

2.2.2 Requisitos de Exploração

1. Permitir a consulta das informações dum Produto.
2. Permitir a consulta das informações dum Cliente.
3. Permitir a consulta das informações duma Encomenda.
4. Permitir a consulta das informações de um Armazém.
5. Permitir a consulta das informações do Veículo Transportador.
6. Listar as Encomendas designadas a um cliente em específico.
7. Listar as Encomendas que decorreram num intervalo de datas.
8. Listar as Encomendas que decorreram numa dada cidade.
9. Listar as Encomendas que foram transportadas por um Veículo Transportador.
10. Listar as Encomendas que têm um determinado produto.
11. Filtrar os Veículos Transportadores que necessitam de ser substituídos, no mês corrente.
12. Filtrar os Veículos Transportadores que necessitam de renovar o seguro, no mês corrente.
13. Filtrar os Veículos Transportadores que necessitam de fazer a inspeção, no mês corrente.
14. Filtrar os Veículos Transportadores que necessitam de renovar o imposto de selo, no mês corrente.
15. Listar Encomendas pendentes.

2.2.3 Requisitos de Controlo

1. Permitir guardar informações relativas a um produto, cliente, encomenda e veículo transportador.
2. Permitir ao funcionário de escritório adicionar um novo cliente.
3. Permitir ao funcionário de escritório atualizar morada de um cliente.
4. Permitir ao funcionário de escritório adicionar um veículo transportador novo.

5. Permitir ao funcionário de escritório atualizar data de renovação do seguro.
6. Permitir ao funcionário de escritório atualizar data da inspeção.
7. Permitir ao funcionário de escritório atualizar data de renovação do imposto de selo.
8. Permitir ao funcionário de escritório atualizar número de quilómetros máximo por encomenda dum veículo transportador.
9. Permitir ao chefe de armazém adicionar um novo produto.
10. Permitir ao funcionário de escritório adicionar uma nova encomenda.
11. Permitir ao motorista adicionar data de entrega e distância da encomenda.
12. Permitir aos gestores da empresa analisar os créditos numa determinada data.

2.3. Perfis de Utilização

O sistema de base de dados a ser desenvolvido, estará na base de uma aplicação *mobile* e de uma aplicação *desktop*. Neste sentido é possível que vários tipos de utilizadores estejam a utilizar concorrentemente o sistema de base de dados.

Existem *à priori* 4 tipos de utilizadores que podem aceder à base de dados, os motoristas, os funcionários de escritório, os gestores da empresa e os chefes de armazém.

A utilização por parte dos motoristas é a mais simples, pois baseando-se num número de identificação da encomenda apenas têm de a validar no sistema.

Os funcionários de escritório têm o objetivo de adicionar possíveis encomendas ou retirá-las caso estas não possam ser concluídas. Podem também atualizar as várias datas relativas à manutenção dos veículos transportadores e adicionar clientes ao sistema de base de dados.

Os gestores empresariais interrogam a base de dados por forma a obterem informações relacionadas com a parte administrativa.

Por último os chefes de armazém, têm a tarefa de adicionar um novo produto à base de dados, caso este não exista, e consultar a base de dados por forma a saber qual o stock de um armazém.

2.4. Análise Geral dos Requisitos

O sistema de base de dados em questão tem como principal utilização auxiliar a gestão e otimização de todas as tarefas às quais a DistribuMinho se compromete.

Assim, a DistribuMinho conseguirá, dentro da própria empresa, consultar informações sobre produtos e clientes, consultar e analisar a sua frota de veículos para, se necessário, fazer as alterações devidas. A empresa terá também controlo sobre o seu balanço financeiro.

Em relação aos clientes, o sistema de base de dados permitirá adicionar e remover clientes, ou apenas atualizar os seus dados e definir qual o veículo utilizado para chegar a um cliente de uma certa zona.

3. Modelação Conceptual

Nesta secção apresentamos os passos efetuados para a construção do modelo conceptual do sistema de base de dados. Inicialmente, descrevemos as entidades envolvidas no modelo e como estas se relacionam entre si. De seguida, identificamos os atributos que caracterizam as diferentes entidades. Para cada atributo, identificamos se este é simples, nulo, derivado, composto ou multivalorado. No final, ilustramos o modelo conceptual desenvolvido e procedemos para a sua validação.

De referenciar que o modelo foi desenvolvido através da ferramenta *TerraER*.

3.1. Apresentação da abordagem de modelação realizada

Tendo por base as necessidades apresentadas no capítulo inicial, tentamos criar o modelo conceptual de forma a fazer transparecer estas ideias.

Considerando que estamos perante um sistema de bases de dados para tratar de distribuições da DistribuMinho, constatámos que uma entidade com muita importância seria uma Encomenda, visto que é através desta que uma distribuição se inicia. Perante tal entidade, determinamos que esta teria uma quantidade de Produtos, visto ser interessante saber o que a Encomenda possui. Estes Produtos são tidos em conta como stock em vários Armazéns da DistribuMinho, de modo que cada Armazém possua os mais variados tipos de stock.

Relacionando agora com a questão da entrega ao cliente, o grupo decidiu estabelecer que cada Armazém teria vários Veículos Transportadores estabelecidos para efetuar o transporte e fazer chegar as Encomendas aos Clientes. Desta forma, os Clientes são considerados como o destino numa rota que se inicia no Armazém a que o Veículo Transportador pertence.

Concluímos, portanto, que o nosso modelo conceptual se baseia em 5 entidades: Encomenda, Produto, Armazém, Veículo Transportador e Cliente.

3.2. Identificação e caracterização das entidades

Para serem definidas de uma forma correta e coesa, as entidades devem apenas identificar os principais objetos que os utilizadores consideram mais interessantes. A extração

destas entidades é feita através de uma análise atenta aos requisitos levantados pelo grupo. Convém ter uma atenção redobrada ao que, em alguns casos, são atributos dos objetos e não entidades.

Seguindo todo este processo, foram escolhidas as seguintes entidades: Veículo Transportador, Armazém, Encomenda, Cliente e Produto.

Caracterização:

- A entidade Veículo Transportador identifica, em geral, um Veículo que está responsável pela entrega de encomendas.
- A entidade Armazém identifica, em geral, um Armazém que contém produtos e onde se encontram os veículos transportadores.
- A entidade Encomenda identifica, em geral, uma Encomenda.
- A entidade Produto identifica, em geral, um Produto que está recolhido em vários armazéns e que pertence a encomendas.
- A entidade Cliente identifica, em geral, um Cliente ao qual está destinada uma encomenda.

3.3. Identificação e caracterização dos relacionamentos

Para identificarmos os relacionamentos existentes no Modelo Conceptual é necessário primeiro encontrar os relacionamentos em si e os verbos que os expressam. Estes relacionamentos têm que estar definidos como requisitos logo, recorrendo à análise destes é possível identificar os relacionamentos entre entidades.

Os relacionamentos são:

- Listar as Encomendas **designadas a** um cliente em específico.
- Listar as Encomendas que foram **transportadas** por um Veículo Transportador.
- Listar as Encomendas que **têm** um determinado produto.
- Listar os Armazéns que **têm** um determinado produto.
- Listar os Armazéns que **têm** um determinado Veículo Transportador.

Existe, portanto, cinco relacionamentos diferentes e para cada um destes é necessário determinar a sua multiplicidade e obrigatoriedade.

- 1) Relacionamento entre as entidades Encomenda e Cliente.
 - ➔ Sabemos que uma Encomenda apenas pode ser designada a um Cliente, mas que um Cliente pode receber uma ou mais Encomendas, logo temos um relacionamento N para 1, em que uma Encomenda está obrigatoriamente associada a um Cliente.
- 2) Relacionamento entre as entidades Veículo Transportador e Encomenda.
 - ➔ Sabemos que um Veículo Transportador pode transportar uma ou mais Encomendas, mas que uma Encomenda apenas pode ser transportada por um Veículo Transportador. Assim temos um relacionamento 1 para N, em que uma Encomenda está obrigatoriamente associada a um Veículo Transportador.
- 3) Relacionamento entre as entidades Encomenda e Produto.
 - ➔ Sabemos que um Produto pode pertencer a uma ou mais Encomendas e uma Encomenda pode ter um ou mais Produtos, não existindo qualquer obrigatoriedade. Concluímos assim que temos um relacionamento de N para M.
- 4) Relacionamento entre as entidades Armazém e Veículo Transportador.
 - ➔ Sabemos que um Armazém pode ter um ou mais Veículos Transportadores, mas um Veículo Transportador só pode pertencer a um Armazém logo, temos um relacionamento de 1 para N, em que um Veículo Transportador está obrigatoriamente associado a um Armazém.
- 5) Relacionamento entre as entidades Armazém e Produto.
 - ➔ Sabemos que um Armazém pode ter um ou mais Produtos e que um Produto pode pertencer a um ou mais Armazéns logo, temos um relacionamento de N para N sem existir qualquer tipo de obrigatoriedade.

3.4. Identificação e associações dos Atributos com as Entidades e Relacionamentos

De seguida, são apresentados todos os atributos que caracterizam as entidades descritas na secção 3.2 e que caracterizam os relacionamentos descritos na secção 3.3.

3.4.1 Armazém

Cada armazém representa a origem de cada encomenda tendo por isso uma morada que é um atributo composto, sendo constituído pelos atributos simples rua e cidade. Como a DistribuMinho é uma empresa que atua unicamente em Portugal é escusado ter um atributo que indique o país a que pertence.

Possui um atributo derivado que indica quantos veículos pertencem ao armazém em causa. Para identificar unicamente esta entidade, cada armazém possui um número de armazém, funcionando assim este atributo como chave primária.

3.4.2 Produto

Um produto é identificado essencialmente por ter um ID que funciona como chave primária, cada produto tem o seu próprio ID. Cada produto é também caracterizado pelo seu nome, pelo seu tipo, podendo atributo tomar valores tais como, charcutaria, lacticínio, peixe, etc. Por último, cada produto tem um preço que pode corresponder ao litro ou ao quilograma.

Todos os atributos desta entidade são atributos simples.

3.4.3 Cliente

Cada cliente possui um NIF que *à priori* é único, pelo que este atributo é simples e funciona como chave primária da entidade. Cada cliente pode ter, ou não, um endereço associado e tem obrigatoriamente um tipo. O atributo tipo pode tomar valores tais como, bar, restaurante, supermercado, minimercado, etc. Tanto o atributo tipo como o atributo endereço são simples.

Tal como acontece na entidade Armazém, cada cliente possui uma morada que é um atributo composto dividido em dois atributos simples, rua e cidade. Repetindo a ideia da entidade Armazém, só são entregues encomendas a clientes residentes em Portugal pelo que não é necessário ter esse atributo presente.

Finalizando, o cliente possui um atributo multivvalorado que é o contacto, logo, a si pode ter vários contactos associados.

3.4.4 Encomenda

A encomenda tem associada a si um identificador único que é atribuído na sua criação, e funciona como chave primária. Esse atributo é o número de encomenda. Cada encomenda é datada aquando o pedido e aquando da entrega, pelo que tem dois diferentes atributos a descrever estas duas situações (data pedido, data entrega). Por último esta entidade possui um atributo distância que descreve os quilómetros efetuados desde a origem até ao destino.

Todos os atributos desta entidade são simples.

3.4.5 Veículo Transportador

Uma vez que cada veículo possui um atributo matrícula, esse é o seu identificador único, pelo que este atributo funciona como chave primária. Tendo que ser sujeito a um certo controlo anualmente, o veículo possui três atributos que permitem aos gestores da empresa

saber quando o veículo tem de ser levado à inspeção (data inspeção) e quando o seguro (data renovação seguro) e o imposto de selo (data renovação imposto selo) têm de ser renovados.

Na DistribuMinho, um condutor dirige apenas um veículo, pelo que cada veículo tem um atributo condutor de forma a saber quem está responsável por cada veículo.

Para esta entidade o SGBD irá guardar informações relacionadas com o fabrico de cada veículo, tendo por isso cada veículo a si associado uma marca e o ano de fabrico. Por último, cada veículo dependendo do seu tipo pode apenas fazer um certo número de quilómetros por deslocação, e possui o número de quilómetros realizados até ao momento.

Todos os atributos desta entidade são simples.

3.4.6 Relacionamento Armazém-Produto

Este relacionamento possui um atributo stock, de forma a poder relacionar qual o stock de cada produto existente em cada Armazém da empresa. Este atributo é simples.

3.4.7 Relacionamento Produto-Encomenda

Este relacionamento possui um atributo quantidade, por forma a saber que quantidade de cada produto será entregue na encomenda. Este atributo é simples.

3.5. Domínio dos Atributos

Por forma a detalhar os diferentes atributos que caracterizam tanto as entidades como os relacionamentos, apresentam-se a seguir duas tabelas que definem o domínio de cada atributo. Associando cada atributo à sua respetiva entidade, ou ao seu respetivo relacionamento.

Para cada atributo é indicado também o tipo de chave que representa, uma breve descrição, se é derivado, se é multivvalorado e ainda se pode ou não possuir um valor nulo.

Entidade	Atributo	Descrição	Domínio	Nulo	Derivado	Multivvalorado
Armazém	Número de Armazém (Chave Primária)	Número identificativo de cada Armazém	Inteiro	Não	Não	Não
	Rua	Rua correspondente à morada do armazém	String com 50 caracteres no máximo	Não	Não	Não

	Cidade	Cidade correspondente à morada do armazém	String com 40 caracteres no máximo	Não	Não	Não
	Número de camiões	Número de camiões que pertence ao armazém	Inteiro	Não	Sim	Não
	ID produto (Chave Primária)	Identificador único de cada produto	Inteiro	Não	Não	Não
Produto	Nome	Nome do produto em causa	String com 50 caracteres no máximo	Não	Não	Não
	Tipo	Tipo do produto	String com 15 caracteres no máximo	Não	Não	Não
	Preço	Preço por unidade ou quilograma	Real	Não	Não	Não
	NIF (Chave Primária)	Número de Identificação Fiscal associado a cada cliente	Inteiro	Não	Não	Não
	Nome	Nome do cliente	String com 50 caracteres no máximo	Não	Não	Não
Cliente	Rua	Rua correspondente à morada do cliente	String com 50 caracteres no	Não	Não	Não

			máximo			
	Cidade	Cidade correspondente à morada do cliente	String com 40 carateres no máximo	Não	Não	Não
	Tipo	Identifica o tipo de estabelecimento do cliente	String com 15 carateres no máximo	Não	Não	Não
	Contacto	Contacto do cliente	Inteiro	Não	Não	Sim
	Endereço	Endereço e-mail de um cliente	String com 40 carateres no máximo	Sim	Não	Não
Encomenda	Número de Encomenda (Chave Primária)	Número identificativo de cada Encomenda	Inteiro	Não	Não	Não
	Data pedido	Data em que o pedido da encomenda foi feito	Data	Não	Não	Não
	Data entrega	Data em que a encomenda foi entregue	Data	Sim	Não	Não
	Distância	Distância percorrida entre a origem e a morada do Cliente	Real	Sim	Não	Não
	Matrícula (Chave Primária)	Identificador único de um veículo	String com 10 carateres no máximo	Não	Não	Não

	Condutor	Funcionário que está responsável pelo Veículo	<i>String</i> com no 40 carateres no máximo	Não	Não	Não
Veículo Transportador	Marca	Marca do Veículo	<i>String</i> com 15 carateres no máximo	Não	Não	Não
	Ano de Fabrico	Ano de fabrico do veículo	Data	Não	Não	Não
	Número de quilómetros realizados	Número total de quilómetros realizados pelo veículo até ao momento	Real	Não	Não	Não
	Número máximo de quilómetros	Número máximo de quilómetros que o veículo pode fazer desde o armazém até ao cliente	Real	Não	Não	Não
	Data de inspeção	Data em que o veículo tem de ser levado à inspeção	Data	Não	Não	Não
	Data de renovação de seguro	Data em que o seguro do veículo tem de ser renovado	Data	Não	Não	Não
	Data de renovação do imposto de selo	Data em que o imposto de selo do veículo tem de ser renovado	Data	Não	Não	Não

Tabela 1 - Domínio dos atributos das entidades

Relacionamento	Atributo	Descrição	Domínio	Nulo	Derivado	Multivalorado
Armazém - Produto	Stock	Stock presente de cada produto num dado armazém	Inteiro	Não	Não	Não
Produto - Encomenda	Quantidade	Quantidade presente dum dado produto numa encomenda	Inteiro	Não	Não	Não

Tabela 2 - Domínio dos atributos dos relacionamentos

3.6. Definição das Chaves Candidatas, Primárias e Alternativas

Para o caso da entidade Armazém podemos definir os atributos *Número de Armazém* e *Rua* como chaves candidatas, visto que ambas definem um único Armazém. Destes dois atributos a *Rua* possui mais caracteres que o *Número de Armazém*, pelo que este último é utilizado como chave primária da entidade Armazém, e a *Rua* permanece como chave alternativa.

Para o caso da entidade Produto, temos como chave candidata o *ID*. O atributo *Nome* não é tido em equação porque é possível dois Produtos possuírem o mesmo *Nome*, o que invalida a escolha deste atributo como chave candidata. Desta forma, o *ID* é estabelecido também como a única chave candidata.

A entidade Cliente possui três chaves candidatas: *NIF*, *Endereço (de e-mail)* e *Contacto*, visto serem únicos para cada pessoa. No entanto, tanto o *Endereço (de e-mail)* como o *Contacto* tem a possibilidade de serem alterados, pelo que o *NIF* é utilizado como chave primária da entidade Cliente, e os restantes atributos referidos permanecem como chaves alternativas.

Para a entidade Encomenda, existem duas chaves candidatas: *Número de Encomenda* e *Data pedido*. No entanto, é possível duas Encomendas serem pedidas na mesma Data, o que torna o *Número de Encomenda* uma escolha mais lógica para chave principal.

Para o caso da entidade Veículo Transportador, temos como chave candidata, e consequente chave primária, a *Matrícula*. A escolha deste atributo é a mais correta, visto não existirem veículos com a mesma *Matrícula*.

3.7. Verificação das Redundâncias no Modelo

Neste subcapítulo, vamos proceder à exameção do modelo conceptual com o objetivo de identificar se existem redundâncias presentes, e caso existam, removê-las.

3.7.1 Reexaminar Relacionamentos (1:1)

Esta verificação direciona-se para o caso em que duas entidades identificam o mesmo objeto no modelo. Visto que o modelo não apresenta qualquer relacionamento do tipo um para um, este passo é ignorado.

3.7.2 Remover Relacionamentos Redundantes

Um relacionamento é redundante se a mesma informação pode ser obtida por via de outros relacionamentos, portanto caso existam, devem ser removidas.

No nosso modelo conceptual não existem redundâncias, isto porque:

- Não existe ligação entre Encomenda e Armazém.
Existindo uma relação entre Encomenda e Veículo Transportador é possível determinar o armazém de onde provém a encomenda, isto porque, um veículo transportador pertence apenas a um armazém e uma encomenda é transportada apenas por um veículo, sendo assim desnecessário uma relação Encomenda-Armazém.

3.7.3 Considerar Dimensão Temporal

Considerar a dimensão temporal consiste em remover relações que futuramente possam deixar de existir devido a qualquer tipo de acontecimento.

No caso do nosso modelo não existe essa preocupação pois existe sempre a necessidade de guardar a informação passada, logo, o fator tempo não é relevante.

3.8. Apresentação e explicação do diagrama ER

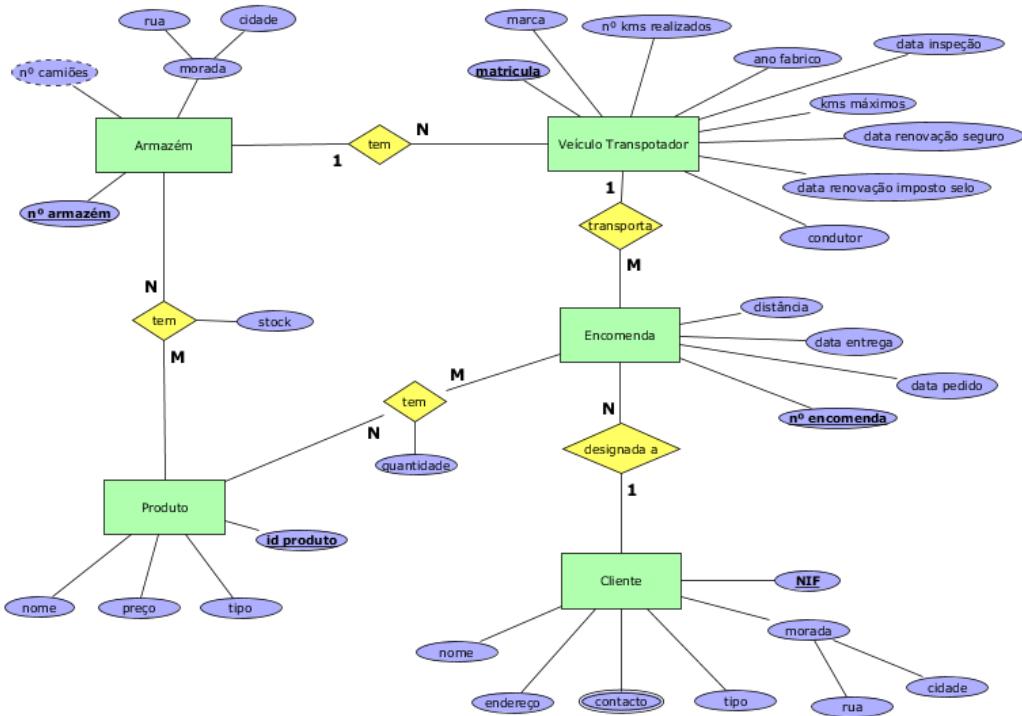


Figura 1 - Modelo Conceptual

A figura acima ilustra o nosso Modelo Conceptual proposto para o sistema de base de dados a desenvolver, seguindo a notação *Chen*. É facilmente visível na figura todas as entidades envolvidas (Armazém, Veículo Transportador, Encomenda, Produto e Cliente). Note-se que não existe nenhuma *weak entity* pois todas as entidades podem ser unicamente identificadas pelo seu atributo chave. Quanto aos relacionamentos é possível verificar que a existência da entidade Encomenda é dependente da entidade Cliente (sem clientes, não existiriam encomendas), é dependente da entidade Veículo Transportador, que por sua vez também depende da entidade Armazém. Assim, o relacionamento Cliente **designa** Encomenda é do tipo *weak relationship*, havendo participação total. Pelo mesmo motivo, as relações Armazém **tem** Veículo Transportador e Veículo Transportador **transporta** Encomenda, são também *weak relationships*, pois se não existissem armazéns, a empresa não teria Veículos Transportadores e se não houvessem Veículos Transportadores, as encomendas não seriam transportadas.

É possível identificar a existência de um atributo multivvalorado, o contacto, um atributo composto, a morada, e um atributo derivado, o número de camiões. O atributo Morada está presente nas entidades Armazém e Cliente.

3.9. Validação do Modelo Conceptual contra Transações do Utilizador

Tendo em conta o sistema de base de dados a desenvolver e, tendo em conta os requisitos que descrevemos na secção 2, consideramos que o Modelo Conceptual deve estar preparado para suportar pelo menos 4 operações fundamentais: adicionar um Cliente, adicionar um Produto, adicionar um Veículo Transportador e realizar uma Encomenda.

3.9.1 Adicionar um novo Cliente

Através do atributo NIF pertencente à entidade Cliente é possível que, ao se adicionar um novo Cliente, se verifique que este já se encontre registado.

3.9.2 Adicionar um novo Produto

Tal como é possível verificar se um Cliente já se encontra registado no sistema ou não, também é possível verificar se um Produto já se encontra registado, ou não, no sistema da empresa.

3.9.3 Adicionar um veículo transportador novo

Através do atributo matrícula, é possível verificar se um veículo transportador se encontra registado ou não, no sistema de base de dados da empresa.

3.9.4 Atualizar informações de um veículo transportador

Através do atributo matrícula, o utilizador poderá aceder a um veículo transportador e atualizar todos os atributos que caracterizam o mesmo como: a data de renovação do seguro, a data de inspeção, a data de renovação do imposto de selo e o número de quilómetros máximo por encomenda desse veículo.

3.9.5 Adicionar um novo produto

Através do atributo idProduto é possível verificar se um Produto já existe no sistema de base dados.

3.9.6 Adicionar uma nova encomenda

Através do atributo NrEncomenda, é possível verificar se uma Encomenda já foi registada no sistema de base dados ou não.

3.9.7 Atualizar a data de entrega da encomenda

Através do atributo NrEncomenda, o utilizador poderá encontrar uma encomenda no sistema e alterar a respetiva data de entrega dessa encomenda.

3.10. Validação do Modelo de Dados com o Utilizador

Tendo em conta o que foi descrito na secção anterior e, tendo em conta os requisitos mínimos que foram impostos, verifica-se que o modelo conceptual construído está preparado para o sistema a desenvolver, pois os utilizadores validaram-no como sendo uma representação real do seu negócio.

4. Modelação Lógica

Nesta secção apresentamos todos os passos, bem detalhados, que foram efetuados na construção do modelo lógico do sistema de base de dados. Inicialmente procedemos à construção e validação do modelo lógico, onde detalhamos a derivação de cada relação partindo do modelo conceitual já exposto na secção 3.9. Depois de concluído este passo, e com ajuda do programa *MySQLWorkbench*, apresentamos o desenho do modelo. De seguida, validamos através das regras de normalização, e verificamos se este era capaz de responder às transações e interrogações estabelecidas. Por último verificamos a integridade do modelo lógico, analisamos o possível crescimento futuro da base de dados e revemos todo o modelo lógico com o utilizador.

4.1. Construção e validação do modelo de dados lógico

O objetivo desta subsecção é criar relações, partindo do modelo conceptual, que representem as entidades, os relacionamentos e os atributos que foram identificados ao longo da secção 3.

Começamos então por especificar o nome das relações e os atributos relacionados a cada uma, baseando-nos nas entidades existentes. De seguida identificamos os tipos das chaves, realçando as mais importantes, as primárias e as estrangeiras.

Passando para os relacionamentos, começamos pelos do tipo **1:N**. Estes tipos de relacionamentos não originam novas relações. Necessitam de atenção pois a chave primária da relação referente à entidade “pai” (entidade do lado do 1), necessita de estar na relação referente à entidade “filho” (entidade do lado do N), como sendo uma chave estrangeira. Os relacionamentos **N:M** dão origem a uma nova relação, que inclui as chaves primárias de ambas as entidades envolvidas e os atributos relacionados a esse relacionamento, caso existam.

Por último, dada a existência de um atributo multivalorado, é necessária a criação de uma nova relação. Esta relação funciona exatamente da mesma forma que um relacionamento **1:N**, onde a relação referente ao atributo multivalorado funciona como entidade “filho” e por isso tem a chave primária da entidade “pai” na sua tabela como sendo uma chave estrangeira.

De referenciar que decidimos adicionar mais duas relações ao modelo lógico para providenciar qualidade de dados, sabendo que em contramão poderíamos ter alguma perda na

eficiência/otimização referente à obtenção desses dados. Esta escolha passou também por se prever que pudessem existir várias entradas numa tabela com o mesmo valor, estando a gastar assim espaço que porventura pode ser necessário futuramente. Essas duas relações são: **Tipo_Produto** e **Tipo_Cliente**.

Na seguinte tabela estão então documentadas, de forma mais sucinta, as relações criadas para modelo lógico.

Armazem (NrArmazem, NrCamioes, rua, cidade)
Chave Primária: NrArmazem
Atributo Derivado: NrCamioes
Produto (idProduto, nome, preco, tipo)
Chave Primária: idProduto
Chave Estrangeira: tipo referencia Tipo_Produto (IdTipoProduto)
Cliente (NIF, nome, rua, cidade, endereço, tipo)
Chave Primária: NIF
Chave Estrangeira: tipo referencia Tipo_Cliente (IdTipoCliente)
Encomenda (NrEncomenda, dataPedido, dataEntrega, distancia, VeiculoTransportador_matricula, Cliente_NIF)
Chave Primária: NrEncomenda
Chave Estrangeira: VeiculoTransportador_matricula referencia VeiculoTransportador (matricula)
Chave Estrangeira: Cliente_NIF referencia Cliente (NIF)
VeiculoTransportador (matricula, marca, NrKmsRealizados, anoFabrico, dataInspecao, kmsMaximos, dataRenovacaoSeguro, dataRenovacaoImpostoSelo, condutor, NrArmazem)
Chave Primária: matricula
Chave Estrangeira: NrArmazem referencia Armazem (NrArmazem)
Armazem_Produto (NrArmazem, idProduto, stock)
Chave Primária: NrArmazem, idProduto
Chave Estrangeira: NrArmazem referencia Armazem (NrArmazem)
Chave Estrangeira: idProduto referencia Produto(idProduto)
Produto_Encomenda (NrEncomenda, idProduto, quantidade)
Chave Primária: NrEncomenda, idProduto
Chave Estrangeira: NrEncomenda referencia Armazem (NrArmazem)
Chave Estrangeira: idProduto referencia Produto(idProduto)
Contacto (contacto, Cliente_NIF)
Chave Primária: contacto
Chave Estrangeira: Cliente_NIF referencia Cliente(NIF)
Tipo_Produto (IdTipoProduto, Designacao)
Chave Primária: IdTipoProduto
Tipo_Cliente (IdTipoCliente, Designacao)
Chave Primária: IdTipoCliente

Tabela 3 - Relações do Modelo Lógico

4.2. Desenho do modelo lógico

Depois de obtermos todas as relações e os respetivos relacionamentos, criamos o modelo lógico. Para isso, utilizamos o MySQL Workbench para desenhar o modelo. O resultado foi o seguinte:

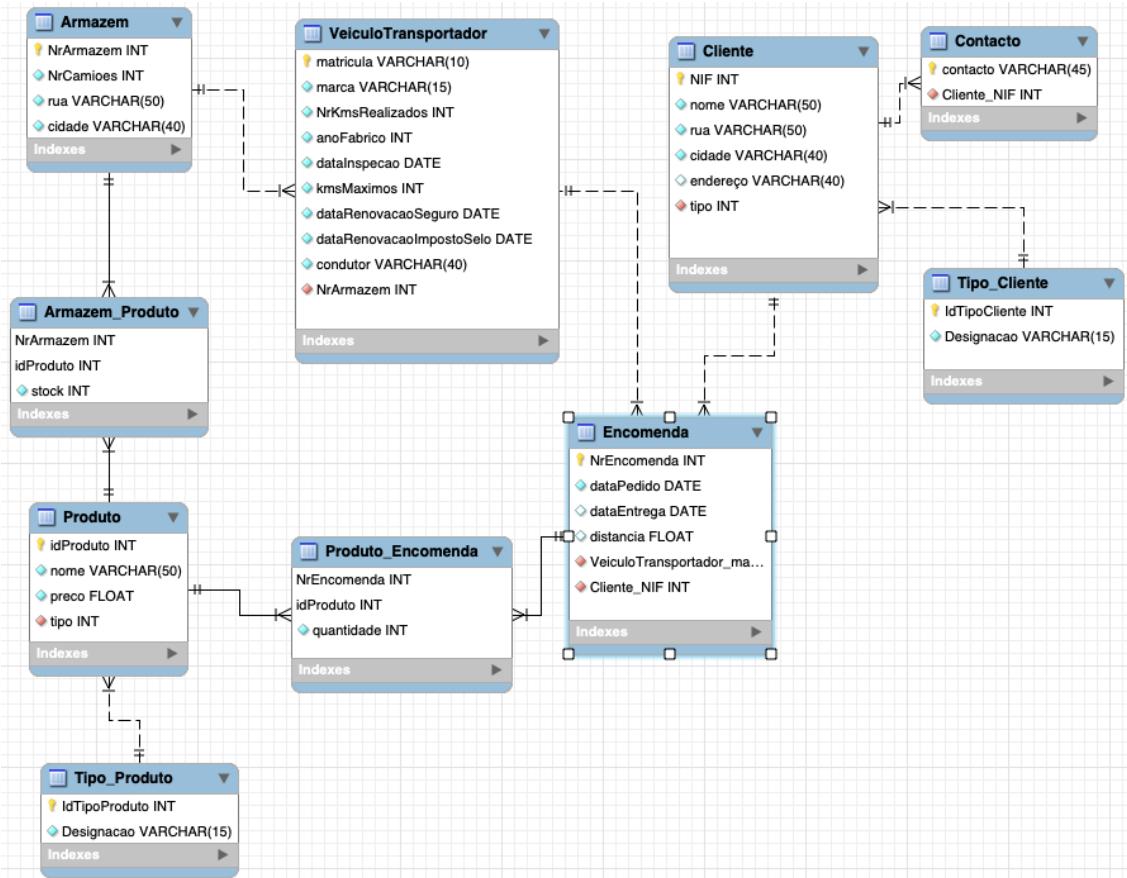


Figura 2 - Modelo Lógico

4.3. Validação do modelo através da normalização

O processo de normalização consiste na produção de relações com as propriedades corretas do nosso modelo, dados os requisitos de uma base de dados com o objetivo de confirmar e aumentar a integridade dos dados que apresentamos e a sua eficiência. Para isso, recorremos a vários métodos de normalização.

Na aplicação destas formas, fizemos um povoamento *ad hoc* do primeiro esboço do nosso modelo lógico e o resultado foi o esquema em seguida apresentado.

<u>NIF</u>	<u>Nome</u>	<u>Rua</u>	<u>Cidade</u>	<u>Endereço</u>	<u>tipo(FK)</u>
2550 00111	José Santos	Rua 31 de Janeiro	Viana do Castelo	jose10santos@gmail.com	1
2551 11222	Restaurante O Pescador	Avenida do Mar	Matosinhos	restauranteopescador@gmail.com	1
2553 33444	Supermercado DaTerra	Rua da Vila	Braga	supermercadodaterra@gmail.com	2

Tabela 4 – Cliente

<u>Contacto</u>	<u>Cliente_NIF (FK)</u>
926555444	255000111
916333222	255111222
966111000	255333444

Tabela 5 – Contacto

<u>IDTipoCliente</u>	<u>Designacao</u>
1	Restaurante
2	Supermercado

Tabela 6 – Tipo_Cliente

<u>NrEncomenda</u>	<u>dataPedido</u>	<u>dataEntrega</u>	<u>Distancia</u>	<u>VeículoTransportador(FK)</u>	<u>Cliente_NIF(FK)</u>
1	19-11-2018	19-11-2018	10	00-RC-10	255000111
2	19-11-2018	19-11-2018	25	15-DB-25	255111222
3	19-11-2018	19-11-2018	50	19-SB-04	255333444

Tabela 7 - Encomenda

<u>Matricula</u>	Marca	nrKmsRealizados	anoFabrico	dataInspecao	kmsMaximos	dataRenovaçãoSeguro	dataRevalidaçãoPoloSelo	Condutor	nrArmazem(FK)
00-RC-10	Toyota	7000	2015	01-12-2017	15000	01-01-2015	01-12-2017	João	1
15-DB-25	Volvo	3000	2017	01-12-2018	20000	01-12-2018	01-12-2017	Manuel	2
19-SB-04	Toyota	500	2018	01-12-2018	300000	01-12-2018	01-12-2017	Augusto	3

Tabela 8 – Veículo Transportador

<u>NrEncomenda(FK)</u>	<u>IdProduto(FK)</u>	quantidade
1	10	100
2	20	200
3	30	300

Tabela 9 - Produto_Encomenda

<u>NrArmazem</u>	<u>NrCamioes</u>	<u>Rua</u>	<u>Cidade</u>
1	1	Rua 31 de Janeiro	Braga
2	1	Avenida do Mar	Porto
3	1	Rua da Vila	Viana do Castelo

Tabela 10 - Armazem

<u>NrArmazem(FK)</u>	<u>IdProduto(FK)</u>	Stock
1	10	1000
2	20	2500
3	30	5000

Tabela 11 - Armazem_Produto

<u>idProduto</u>	Nome	Preco	tipo(FK)
10	Robalo	5	1
20	Arroz	1	1
30	Vinho do Porto	10	2

Tabela 12 - Produto

<u>idTipoProduto</u>	Designacao
1	Alimentação
2	Bebida

Tabela 13 - Tipo_Produto

4.3.1 *Unnormalized Form (UNF)*

Segundo esta regra, **uma tabela contém um ou mais grupos repetidos**. Visualizando o nosso modelo, é possível reparar que esta norma não se verifica.

4.3.2 *First Normal Form (1NF)*

Segundo esta regra, **a interseção entre cada linha e cada coluna deve conter um e um só valor**. Ora, pela simples observação de todas as tabelas se percebe que o esquema cumpre a regra.

4.3.3 *Second Normal Form (2NF)*

Todos os atributos de uma tabela devem ser totalmente funcionalmente dependentes da chave primária dessa tabela.

Como é possível observar pelas tabelas, o nosso esquema cumpre esta regra pois são todos os atributos são, funcionalmente, dependentes a 100% das chaves primárias das tabelas que pertencem.

4.3.4 Third Normal Form (3NF)

Uma relação que cumpre a primeira e a segunda regras de normalização e em que todos os seus atributos, que não sejam chaves primárias, não são transitivamente dependentes da chave primária.

Através da visualização e compreensão das tabelas presentes no nosso Modelo Lógico, é possível verificar que todos os atributos presentes são determinados apenas através da chave primária e não de outra forma possível. Conclui-se, pois, que o modelo por nós apresentado cumpre a regra número três da normalização.

4.4. Validação do modelo com interrogações do utilizador

Para este tipo de validação é necessário provar que, a partir do modelo lógico, é possível responder às interrogações do utilizador, que para todos os efeitos são requisitos de exploração.

4.4.1 Permitir a consulta das informações de um Produto.

As informações relativas a um **Produto** são obtidas através da observação da relação **Produto** e da relação **Tipo_Produto**. A relação **Produto** indica-nos informações tais como: preço e nome. Na relação **Tipo_Produto** podemos obter a sua designação.

4.4.2 Permitir a consulta das informações dum Cliente.

As informações relativas a um **Cliente** são obtidas através da observação da relação **Cliente**, **Contacto** e da relação **Tipo_Cliente**. A relação **Cliente** indica-nos informações tais como: nome, rua e cidade. Na relação **Tipo_Cliente** podemos encontrar a designação de um Cliente e na relação **Contacto**, podemos obter um ou mais contactos do Cliente em questão.

4.4.3 Permitir a consulta de informações duma Encomenda.

As informações relativas a uma **Encomenda** são obtidas através da observação da relação **Encomenda**. A relação **Encomenda** indica-nos informações tais como: distância, data de entrega e data de pedido.

4.4.4 Permitir a consulta de informações de um Armazém.

As informações relativas a um **Armazém** são obtidas através da observação da relação **Armazém**. A relação Armazém indica-nos informações tais como: número de camiões, rua e cidade.

4.4.5 Permitir a consulta das informações do Veículo Transportador.

As informações de um **Veículo Transportador** são obtidas através da observação da relação **VeículoTransportador**. Nesta relação estão presentes informações tais como: marca, ano de fabrico, número de quilómetros realizados e data de inspeção, por exemplo.

4.4.6 Listar as Encomendas designadas a um cliente em específico.

A relação **Encomenda** tem como chave estrangeira o NIF de um **Cliente**, o que nos permite obter quais as encomendas designadas ao cliente que possui o NIF procurado pelo utilizador.

4.4.7 Listar as Encomendas que decorrem num intervalo de datas.

A relação **Encomenda** tem informações relativas à data de pedido e de entrega de uma encomenda específica. Assim, sabendo o intervalo de datas basta filtrar as encomendas para que apenas fiquem listadas as encomendas que pertencem a esse intervalo de datas.

4.4.8 Listar as Encomendas que decorreram numa dada cidade.

A relação **Encomenda** tem informações relativas ao NIF de um **Cliente**, ao qual uma encomenda específica foi designada. A partir daqui, basta aceder às informações do **Cliente** em causa e comparar se a cidade deste é igual à cidade pretendida pelo utilizador.

4.4.9 Listar as Encomendas que foram transportadas por um Veículo Transportador.

A relação **Encomenda** tem como chave estrangeira a matrícula do veículo transportador de uma encomenda específica logo, é possível obter as Encomendas que foram transportadas pelo Veículo procurado pelo utilizador.

4.4.10 Listar as Encomendas que têm um determinado Produto.

A relação **Produto_Encomenda** tem informações relativas ao produto que está a ser transportado, nomeadamente a chave primária idProduto, uma vez que esta associa um produto a uma encomenda.

4.4.11 Informar sobre o balanço crédito numa determinada data.

A relação **Encomenda** tem informação relativa à data de Entrega de uma encomenda em específico, logo, acedendo à informação relativa à relação **Produto_Encomenda** através do identificador da **Encomenda** (NrEncomenda) podemos retirar o(s) produto(s) presente(s) na **Encomenda**.

De seguida, é possível, obter o(s) preço(s) do(s) produto(s) em causa. Multiplicando este valor pela quantidade do(s) produto(s) (presente(s) na relação **Produto_Encomenda**) obtemos o valor da **Encomenda** em causa.

Por fim, basta somar todos os valores das encomendas entregues antes da data determinada pelo utilizador e obtemos o balanço financeiro nessa data.

4.4.12 Filtrar os Veículos Transportadores que necessitam de ser substituídos, no mês corrente.

Na relação **VeículoTransportador** é possível encontrar informação relativa ao número de quilómetros realizados. Assim basta filtrar os veículos que têm um número de quilómetros realizados superior ao máximo imposto pela gerência, significando assim que têm que ser substituídos.

4.4.13 Filtrar os Veículos Transportadores que necessitam de fazer a inspeção, no mês corrente.

Na relação **VeículoTransportador** é possível encontrar informação relativa à data de inspeção. Assim basta filtrar os veículos que têm data de inspeção iguais ao mês e ano correntes.

4.4.14 Filtrar os Veículos Transportadores que necessitam de renovar o imposto de selo, no mês corrente.

Na relação **VeículoTransportador** é possível encontrar informação relativa à data de renovação de imposto de selo. Assim basta filtrar os veículos que têm data de renovação de imposto de selo iguais ao mês e ano correntes.

4.4.15 Filtrar os Veículos Transportadores que necessitam de renovar o seguro, no mês corrente.

Na relação **VeículoTransportador** é possível encontrar informação relativa à data de renovação de seguro. Assim basta filtrar os veículos que têm data de renovação do seguro igual ao mês e ano correntes.

4.5. Validação do modelo com as transações estabelecidas

4.5.1 Adicionar um novo Veículo Transportador

Caso a matrícula ainda não exista, insere-se a matrícula, número de quilómetros realizados, marca, ano de fabrico, data de inspeção, quilómetros máximos por encomenda, data de renovação do seguro, data de renovação do imposto de selo, o condutor e o número de armazém ao qual o veículo transportador pertence na tabela **VeiculoTransportador**.

Como a matrícula (chave primária do Veículo Transportador) está presente (como chave estrangeira) na tabela **Encomenda**, é possível relacionar **Encomendas** com os seus respetivos **Veículos Transportadores**.

Quando é adicionado um novo veículo transportador, insere-se o número do armazém ao qual este pertence, logo, o valor **NrCamioes** vai ser incrementado.

4.5.2 Adicionar um novo Cliente

Caso o NIF ainda não exista, insere-se o NIF, o nome, a rua, a cidade, o endereço e tipo na tabela Cliente.

Como o NIF (chave primária do Cliente) está presente (como chave estrangeira) na tabela Encomenda, é possível relacionar Encomendas com os seus respetivos Clientes. O mesmo acontece para a tabela Contacto, logo, também é possível relacionar o Cliente com o(s) seu(s) respetivo(s) contacto(s).

4.5.3 Adicionar uma nova Encomenda

Insere-se a data de pedido, a distância, o Veículo Transportador e o Cliente da encomenda na tabela Encomenda. A chave primária NrEncomenda é automaticamente incrementada.

Como o NrEncomenda (chave primária da Encomenda) está presente (como chave estrangeira) na tabela intermédia Produto_Encomenda, é possível relacionar Encomendas com Produtos.

4.5.4 Adicionar um novo Produto

Insere-se o nome, o preço e o tipo do produto na tabela Produto. A chave primária idProduto é automaticamente incrementada.

Como já foi referido no ponto anterior, é possível relacionar Encomendas com Produtos, estando também presente na tabela Produto_Encomenda (como chave estrangeira) o idProduto, que é a chave primária do Produto.

4.5.5 Adicionar um novo Tipo de Produto

Insere-se a designação do tipo do produto na tabela Tipo_Produto. A chave primária idTipoProduto é automaticamente incrementada.

Como o idTipoProduto (chave primária do tipo de produto) está presente (como chave estrangeira, designada por tipo) na tabela Produto, é possível relacionar Produto com o seu tipo respetivo.

4.5.6 Adicionar um novo Tipo de Cliente

Insere-se a designação do tipo do cliente na tabela Tipo_Cliente. A chave primária idTipoCliente é automaticamente incrementada.

Como o idTipoCliente (chave primária do tipo de cliente) está presente (como chave estrangeira, designada por tipo) na tabela Cliente, é possível relacionar Cliente com o seu tipo respetivo.

4.5.7 Adicionar um novo Contacto

Caso o contacto ainda não exista, insere-se o contacto e o NIF do Cliente a quem pertence o contacto na tabela Contacto.

4.5.8 Indicar que um Produto está presente numa Encomenda

As tabelas Produto e Encomenda estão associadas entre si na tabela intermédia Produto_Encomenda. Esta tabela possui duas chaves estrangeiras – NrEncomenda e idProduto, que são as chaves primárias das tabelas Encomenda e Produto, respetivamente. Também possui um atributo denominado de quantidade que representa a quantidade do produto que está a ser transportada na encomenda.

Quando esta transação é estabelecida é retirada a quantidade do produto que foi transportado na encomenda ao seu respetivo *stock*. Esta ação é possível pois, utilizando a chave estrangeira idProduto juntamente com a chave estrangeira NrArmazem podemos aceder à tabela Armazem_Produto e atualizar o valor do *stock*.

A chave idProduto é facilmente retirada pois pertence também à tabela intermédia Produto_Encomenda. No caso da chave NrArmazem, é necessário aceder à tabela Encomenda através da chave estrangeira NrEncomenda e determinar qual o veículo que realizou a encomenda. Posteriormente é necessário utilizar a chave VeiculoTransportador_matricula (presente na tabela Encomenda) para aceder à tabela VeiculoTransportador para finalmente poder determinar o Armazém ao qual este pertence.

4.5.9 Indicar que um Produto está recolhido num Armazém

As tabelas Produto e Armazém estão associadas entre si na tabela Armazem_Produto. Esta tabela possui duas chaves estrangeiras – idProduto e NrArmazem, que são as chaves primárias das tabelas Produto e Armazém respetivamente. Também possui um atributo denominado de *stock* que representa a quantidade de produto que está recolhida no armazém.

4.6. Verificação da integridade das restrições

Nesta subsecção apresentamos as restrições que desejamos que se cumpram de forma a proteger o nosso sistema de base de dados. Deste modo visamos que problemas como a inconsistência, imprecisão e dados incompletos sejam extinguidos.

O objetivo desta análise de integridade de restrições baseia-se na necessidade de oferecer à empresa uma representação “verdadeira” dos seus requisitos de dados.

4.6.1 Dados Necessários

Existem no nosso modelo lógico 3 atributos que podem ser *NULL*, 2 atributos que estão na relação Encomenda, e 1 que está na relação Cliente. No caso dos atributos da relação Encomenda (*distança*, *dataEntrega*), estes podem ser nulos uma vez que são atributos de “confirmação de entrega”. Apenas são adicionados quando a encomenda for entregue e o motorista adicionar essa informação. Quanto ao atributo da relação Cliente (*endereço*), este pode ser nulo uma vez que um cliente não tem necessariamente de ter um endereço de e-mail.

Todos os outros atributos que não os referidos não podem ser nulos, tal como já referido na secção 3.5. Desta forma no nosso modelo lógico todos estão assinalados com a opção *NOT NULL*.

4.6.2 Domínio das Restrições dos Atributos

O domínio de cada atributo foi escolhido especificamente para as necessidades de cada um, contudo as decisões que foram tomadas em relação a estes domínios foram já descritas na secção 3.5. Para valores inteiros, os atributos da nossa base de dados possuem o tipo de dados INT, e para atributos como o preço ou a distância usamos o tipo de dados FLOAT. Para atributos cujos valores são *Strings*, optamos por usar VARCHAR com o tamanho a variar consoante cada atributo. Por último, para atributos que se baseiam em datas optamos por usar DATE porque não nos interessa saber a hora específica da entrega ou do pedido.

4.6.3 Cardinalidade

As restrições associadas ao relacionamento entre dados do nosso sistema de base de dados foram já introduzidas na secção 3.3, e levemente abordadas na secção 4.1. De uma forma muito simples o nosso modelo têm 2 relacionamentos do tipo **N:M**. Uma Encomenda pode ter muitos Produtos, e por sua vez um Produto pode estar em várias Encomendas. Por outro lado, um Armazém tem vários produtos e um Produto pode estar em muitos Armazéns. Como já foi referido anteriormente este tipo de relacionamentos deram origem a duas tabelas intermediárias no modelo lógico **Armazem_Produto** e **Produto_Encomenda**.

Existem 3 relacionamentos do tipo **1:N**, mas de forma diferente dos relacionamentos **N:M**, estes não originam novas tabelas apenas exigem que a chave primária da relação “pai”, esteja presente na relação “filho”, como sendo chave estrangeira. Desta forma podemos constatar que um Armazém tem vários Veículos Transportadores, mas um Veículo Transportador apenas pertence a um Armazém. Um Veículo Transportador realiza várias Encomendas, mas uma Encomenda apenas é transportada por um Veículo. Por último, um Cliente pode ter designadas a si várias Encomendas, contudo uma Encomenda é destinada apenas a um Cliente.

4.6.4 Integridade das Entidades

No nosso modelo lógico nenhuma relação pode ter uma chave primária cujo valor seja nulo. Assinalámos em todas as chaves primárias a opção *NOT NULL* para que esta restrição fosse cumprida.

4.6.5 Integridade Referencial

Como já abordado na subsecção 4.6.3, existem 3 relacionamentos do tipo **1:N**, e por isso é necessário ter em atenção alguns casos para que a integridade referencial não seja violada quando se alteram dados que fazem referência a outras tabelas, ou são referenciados por outras tabelas. De um modo mais simples é necessário ter cuidado quando se faz alguma operação que possa envolver referencias, por exemplo através de chaves estrangeiras, a outras tabelas.

Na operação de inserção de alguma entrada nas relações: VeiculoTransportador, Encomenda, Produto_Encomenda e Armazem_Produto, é preciso ter atenção uma vez que estas são relações do tipo “filho”, porque não se deve inserir valores nestas tabelas se a referência à relação “pai” for nula. Apenas se deve proceder à inserção de valores quando existir uma relação “pai” válida.

Na operação de remoção de entradas de relações do tipo “pai”, é necessário ter certos cuidados uma vez que as relações do tipo “filho” possuem referencias para estas tabelas. Remover uma entrada de uma relação do tipo “pai” violaria a integridade referencial uma vez que a chave estrangeira presente na relação “filho” passaria a ter um valor nulo. Estes casos podem então ser tratados através de uma ação conhecida como *CASCADE*, que dita que sempre que haja uma remoção de uma entrada de uma relação “pai”, todas as entradas de relações do tipo “filho” são também removidas mantendo a integridade referencial. No caso do nosso modelo lógico se removêssemos um Cliente, teríamos também de remover as entradas de todas as encomendas que se referissem a esse Cliente, e ainda todas as entradas da relação Produto_Encomenda referentes a todas às encomendas removidas. Seria ainda necessário remover todos os contactos existentes que se referissem a este Cliente.

4.6.6 Restrições Gerais

Existem certos atributos das entidades, para além das chaves primárias, que conseguem identificar a entidade como sendo única. Esses atributos merecem um destaque especial uma vez que podem ser marcados com a opção *UNIQUE*, disponibilizada pelo MySQLWorkBench e passam também a ser *INDEX*.

Exemplos deste caso são: o atributo designação do Tipo_Produto e do Tipo_Cliente uma vez que estas designações têm de ser únicas a cada produto e cliente, não seria coerente dois Tipo_Produto terem chaves primárias diferentes e designações iguais. Tal também acontece quanto ao nome de um Produto e um endereço de um Cliente.

Uma vez que existem atributos que são do tipo *INT* e *FLOAT*, é necessário torná-los positivos, pois no nosso sistema não existem atributos do tipo numérico que possam ser negativos, seria incoerente se tal situação ocorresse. Para marcar este facto, neste tipo de atributos ativamos a opção *UNSIGNED*.

Alguns casos deste tipo são: o número de camiões de um Armazém, o preço de um Produto, o número de quilómetros de um Veículo transportador, etc.

Alguns atributos, discriminando por exemplo o Id de um Produto, o Número de uma Encomenda, o Número de um Armazém, o Id de um tipo de cliente e o Id do tipo do produto, não são inseridos diretamente por quem está a inserir dados deste género no sistema. Não seria nada cómodo ter que saber qual é o id que está livre a seguir para um Produto, logo a solução para estes casos é acionar a opção *AUTO INCREMENT*, por forma a que na inserção deste tipo de dados, estes atributos sejam gerados automaticamente.

4.7. Revisão do modelo lógico com o utilizador

Os utilizadores são requeridos para rever o modelo lógico, de modo a confirmar se consideram que estão perante uma representação fiável dos dados necessários para o bom funcionamento da ideia em causa.

Este modelo permite o registo e manutenção de Clientes e de Encomendas, realizado pelos funcionários de escritório. Permite também o registo e manutenção de Produtos, sendo tal realizado pelos chefes de armazém.

Como foi descrito anteriormente, o Veículo Transportador tem também uma influência grande no bom funcionamento da DistribuMinho. Desta forma, toda a gestão destes está também a cargo dos funcionários de escritório. Os motoristas, que conduzem os Veículos, são responsáveis pela confirmação da entrega duma Encomenda.

Concluímos, portanto, que o modelo apresentado cumpre os requisitos impostos para uma boa utilização dos utilizadores em causa.

4.8. Verificar futuro crescimento

Um bom modelo lógico deve ser capaz de ser estendido para suportar possíveis desenvolvimentos futuros. Desta forma, o modelo terá capacidade para suportar novos requisitos dos utilizadores.

Como o sistema em questão se encontra normalizado e sem redundâncias, é de concluir que será mais acessível a sua manutenção e futuro desenvolvimento. Além disto, o modelo encontra-se capaz de suportar aumentos a todos os níveis, como por exemplo, nas Encomendas, nos Produtos ou nos Armazéns. Um exemplo deste aumento seria, por exemplo, a construção de um novo armazém numa zona diferente de forma a aumentar a área de negócio, ou mesmo expandir a DistribuMinho além-fronteiras.

A forma como as relações presentes estão relacionadas permite que novos requisitos sejam suportados, e desta forma, não existam incompatibilidades ao nível dos mesmos.

5. Implementação Física

Neste capítulo vamos abordar a construção do modelo físico. Vamos também detalhar a passagem do modelo lógico para o modelo físico. Para este processo utilizamos a ferramenta *MySQL*. É nesta fase que serão realizadas as traduções de interrogações do utilizador, as transações estabelecidas e as vistas de utilização.

5.1. Seleção do sistema de gestão de bases de dados

Para a gestão de Base de Dados optámos pelo *MySQL*. Esta decisão foi tomada tendo em conta vários parâmetros, sendo estes:

- A familiarização da ferramenta por parte de todos os elementos do grupo devido à utilização desta nas aulas lecionadas na disciplina de Base de Dados.
- A propriedade transacional do *MySQL*, respeitando:
 - ➔ a Atomicidade – todas as ações realizadas só são efetivadas se forem concluídas com sucesso (princípio de *rollback* em caso de erro e de *commit* em caso de sucesso).
 - ➔ a Consistência – todas as regras/restricções impostas no SGBD devem ser obedecidas.
 - ➔ o Isolamento – cada transação funciona à parte de outras transações, ou seja, nenhuma outra transação pode interferir na transação corrente.
 - ➔ a Durabilidade – todos os resultados de uma transação são permanentes e só podem ser desfeitos por uma ação subsequente.

5.2. Tradução do esquema lógico para o sistema de gestão de bases de dados escolhido em SQL

Tendo presente o modelo lógico presente na figura 2, é realizada a tradução para o SGBD da DistribuMinho através do *Forward Engineer* disponibilizado pelo *Database*. Cada relação do modelo lógico dá origem a uma tabela no SGBD. Inicialmente é criado o *Schema* da DistribuMinho, e depois todas as relações presentes no modelo lógico.

```
-- Schema DistribuMinho
-- 
CREATE SCHEMA IF NOT EXISTS `DistribuMinho` DEFAULT CHARACTER SET utf8 ;
USE `DistribuMinho` ;
```

Figura 3 - Create Schema DistribuMinho

Uma Schema é uma coleção de objetos de uma base de dados que, de alguma forma, estão relacionados entre si. Estes objetos podem ser *tables*, *views*, *translations*, etc.

Uma *Table* de um determinado relacionamento descreve todos os atributos presentes, a chave primária, os índices e as chaves estrangeiras relativas às entidades que estas se relacionam. Tal pode ser verificado nas figuras seguintes.

Desta forma, é possível realizar uma boa tradução do modelo lógico para o SGBD em questão.

```
-- Table `DistribuMinho`.`Produto`

CREATE TABLE IF NOT EXISTS `DistribuMinho`.`Produto` (
  `idProduto` INT NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(50) NOT NULL,
  `preco` FLOAT UNSIGNED NOT NULL,
  `tipo` INT NOT NULL,
  PRIMARY KEY (`idProduto`),
  INDEX `Tipo_Produto_idx` (`tipo` ASC) VISIBLE,
  UNIQUE INDEX `nome_UNIQUE` (`nome` ASC) VISIBLE,
  CONSTRAINT `fk_Tipo_Produto`
    FOREIGN KEY (`tipo`)
    REFERENCES `DistribuMinho`.`Tipo_Produto` (`IdTipoProduto`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

Figura 5 - Create Table Produto

```
-- Table `DistribuMinho`.`Encomenda`

CREATE TABLE IF NOT EXISTS `DistribuMinho`.`Encomenda` (
  `NrEncomenda` INT NOT NULL,
  `dataPedido` DATE NOT NULL,
  `dataEntrega` DATE NULL,
  `distancia` FLOAT UNSIGNED NULL,
  `VeiculoTransportador_matricula` VARCHAR(10) NOT NULL,
  `Cliente_NIF` INT NOT NULL,
  PRIMARY KEY (`NrEncomenda`),
  INDEX `fk_Encomenda_VeiculoTransportador1_idx` (`VeiculoTransportador_matricula` ASC),
  INDEX `fk_Encomenda_Cliente1_idx` (`Cliente_NIF` ASC),
  CONSTRAINT `fk_Encomenda_VeiculoTransportador1`
    FOREIGN KEY (`VeiculoTransportador_matricula`)
    REFERENCES `DistribuMinho`.`VeiculoTransportador` (`matricula`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Encomenda_Cliente1`
    FOREIGN KEY (`Cliente_NIF`)
    REFERENCES `DistribuMinho`.`Cliente` (`NIF`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

Figura 4 - Create Table Encomenda

```

-- Table `DistribuMinho`.`Armazem`

CREATE TABLE IF NOT EXISTS `DistribuMinho`.`Armazem` (
  `NrArmazem` INT NOT NULL AUTO_INCREMENT,
  `NrCamioes` INT UNSIGNED NOT NULL,
  `rua` VARCHAR(50) NOT NULL,
  `cidade` VARCHAR(40) NOT NULL,
  PRIMARY KEY (`NrArmazem`)
)
ENGINE = InnoDB;

```

Figura 8 - Create Table Armazem

```

-- Table `DistribuMinho`.`Tipo_Cliente`

CREATE TABLE IF NOT EXISTS `DistribuMinho`.`Tipo_Cliente` (
  `IdTipoCliente` INT NOT NULL AUTO_INCREMENT,
  `Designacao` VARCHAR(15) NOT NULL,
  PRIMARY KEY (`IdTipoCliente`),
  UNIQUE INDEX `Designacao_UNIQUE` (`Designacao` ASC) VISIBLE
)
ENGINE = InnoDB;

```

Figura 7 - Create Table Tipo_Cliente

```

-- Table `DistribuMinho`.`Tipo_Produto`

CREATE TABLE IF NOT EXISTS `DistribuMinho`.`Tipo_Produto` (
  `IdTipoProduto` INT NOT NULL AUTO_INCREMENT,
  `Designacao` VARCHAR(15) NOT NULL,
  PRIMARY KEY (`IdTipoProduto`),
  UNIQUE INDEX `Designacao_UNIQUE` (`Designacao` ASC) VISIBLE
)
ENGINE = InnoDB;

```

Figura 6 - Create Table Tipo_Produto

```

-- Table `DistribuMinho`.`Cliente`

CREATE TABLE IF NOT EXISTS `DistribuMinho`.`Cliente` (
  `NIF` INT NOT NULL,
  `nome` VARCHAR(50) NOT NULL,
  `rua` VARCHAR(50) NOT NULL,
  `cidade` VARCHAR(40) NOT NULL,
  `endereco` VARCHAR(40) NULL,
  `tipo` INT NOT NULL,
  PRIMARY KEY (`NIF`),
  INDEX `fk_Tipo_Cliente_idx` (`tipo` ASC) VISIBLE,
  UNIQUE INDEX `endereco_UNIQUE` (`endereco` ASC) VISIBLE,
  CONSTRAINT `fk_Tipo_Cliente`
    FOREIGN KEY (`tipo`)
    REFERENCES `DistribuMinho`.`Tipo_Cliente` (`IdTipoCliente`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;

```

Figura 10 - Create Table Cliente

```

-- Table `DistribuMinho`.`Contacto`

CREATE TABLE IF NOT EXISTS `DistribuMinho`.`Contacto` (
  `contacto` VARCHAR(45) NOT NULL,
  `Cliente_NIF` INT NOT NULL,
  PRIMARY KEY (`contacto`),
  INDEX `fk_Contacto_Cliente1_idx` (`Cliente_NIF` ASC) VISIBLE,
  CONSTRAINT `fk_Contacto_Cliente1`
    FOREIGN KEY (`Cliente_NIF`)
    REFERENCES `DistribuMinho`.`Cliente` (`NIF`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;

```

Figura 9 - Create Table Contacto

```

-- Table `DistribuMinho`.`VeiculoTransportador`

CREATE TABLE IF NOT EXISTS `DistribuMinho`.`VeiculoTransportador` (
  `matricula` VARCHAR(10) NOT NULL,
  `marca` VARCHAR(15) NOT NULL,
  `NrKmsRealizados` INT UNSIGNED NOT NULL,
  `anoFabrico` INT UNSIGNED NOT NULL,
  `dataInspeccao` DATE NOT NULL,
  `kmsMaximos` INT UNSIGNED NOT NULL,
  `dataRenovacaoSeguro` DATE NOT NULL,
  `dataRenovacaoImpostoSelo` DATE NOT NULL,
  `condutor` VARCHAR(40) NOT NULL,
  `NrArmazem` INT NOT NULL,
  PRIMARY KEY (`matricula`),
  INDEX `fk_VeiculoTransportador_Armazem1_idx` (`NrArmazem` ASC) VISIBLE,
  CONSTRAINT `fk_VeiculoTransportador_Armazem1`
    FOREIGN KEY (`NrArmazem`)
    REFERENCES `DistribuMinho`.`Armazem` (`NrArmazem`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;

```

Figura 12 - Create Table VeiculoTransportador

```

-- Table `DistribuMinho`.`Encomenda`

CREATE TABLE IF NOT EXISTS `DistribuMinho`.`Encomenda` (
  `NrEncomenda` INT NOT NULL,
  `dataPedido` DATE NOT NULL,
  `dataEntrega` DATE NULL,
  `distancia` FLOAT UNSIGNED NULL,
  `VeiculoTransportador_matricula` VARCHAR(10) NOT NULL,
  `Cliente_NIF` INT NOT NULL,
  PRIMARY KEY (`NrEncomenda`),
  INDEX `fk_Encomenda_VeiculoTransportador1_idx` (`VeiculoTransportador_matricula` ASC) VISIBLE,
  INDEX `fk_Encomenda_Cliente1_idx` (`Cliente_NIF` ASC) VISIBLE,
  CONSTRAINT `fk_Encomenda_VeiculoTransportador1`
    FOREIGN KEY (`VeiculoTransportador_matricula`)
    REFERENCES `DistribuMinho`.`VeiculoTransportador` (`matricula`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Encomenda_Cliente1`
    FOREIGN KEY (`Cliente_NIF`)
    REFERENCES `DistribuMinho`.`Cliente` (`NIF`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;

```

Figura 11 - Create Table Encomenda

```

-- Table `DistribuMinho`.`Produto_Encomenda`

CREATE TABLE IF NOT EXISTS `DistribuMinho`.`Produto_Encomenda` (
  `NrEncomenda` INT NOT NULL,
  `idProduto` INT NOT NULL,
  `quantidade` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`NrEncomenda`, `idProduto`),
  INDEX `fk_Produto_Encomenda_Produto1_idx` (`idProduto` ASC) VISIBLE,
  INDEX `fk_Produto_Encomenda_Encomenda1_idx` (`NrEncomenda` ASC) VISIBLE,
  CONSTRAINT `fk_Produto_Encomenda_Encomenda1`
    FOREIGN KEY (`NrEncomenda`)
      REFERENCES `DistribuMinho`.`Encomenda` (`NrEncomenda`)
      ON DELETE CASCADE
      ON UPDATE CASCADE,
  CONSTRAINT `fk_Produto_Encomenda_Produto1`
    FOREIGN KEY (`idProduto`)
      REFERENCES `DistribuMinho`.`Produto` (`idProduto`)
      ON DELETE CASCADE
      ON UPDATE CASCADE)
ENGINE = InnoDB;

```

Figura 14 - Create Table Produto_Encomenda

```

-- Table `DistribuMinho`.`Armazem_Produto`

CREATE TABLE IF NOT EXISTS `DistribuMinho`.`Armazem_Produto` (
  `NrArmazem` INT NOT NULL,
  `idProduto` INT NOT NULL,
  `stock` INT NOT NULL,
  PRIMARY KEY (`NrArmazem`, `idProduto`),
  INDEX `fk_Armazem_Produto_Produto1_idx` (`idProduto` ASC) VISIBLE,
  INDEX `fk_Armazem_Produto_Armazem1_idx` (`NrArmazem` ASC) VISIBLE,
  CONSTRAINT `fk_Armazem_Produto_Armazem1`
    FOREIGN KEY (`NrArmazem`)
      REFERENCES `DistribuMinho`.`Armazem` (`NrArmazem`)
      ON DELETE CASCADE
      ON UPDATE CASCADE,
  CONSTRAINT `fk_Armazem_Produto_Produto1`
    FOREIGN KEY (`idProduto`)
      REFERENCES `DistribuMinho`.`Produto` (`idProduto`)
      ON DELETE CASCADE
      ON UPDATE CASCADE)
ENGINE = InnoDB;

```

Figura 13 - Create Table Armazem_Produto

5.3. Tradução das interrogações do utilizador para SQL

5.3.1 Permitir a consulta das informações dum Cliente

Esta query é utilizada quando um utilizador, que possua as permissões para tal, pretende ter conhecimento das informações dum cliente.

```
CREATE PROCEDURE InformacoesDeUmCliente (IN NIF int)
SELECT *
FROM Cliente
WHERE Cliente.NIF = NIF;
```

Figura 15 - Procedure InformacoesDeUmCliente

O utilizador fornece o NIF do cliente que pretende saber as informações, e estas são todas apresentadas.

5.3.2 Listar as Encomendas que decorreram num intervalo de datas

Esta query é utilizada quando um gestor de armazém pretende listar as encomendas que decorreram num determinado intervalo de datas.

```
CREATE PROCEDURE EncomendasNumIntervaloDatas(IN dataInicio DATE, IN dataFim DATE)
SELECT e.NrEncomenda, e.DataPedido, e.DataEntrega, c.Nome
FROM Encomenda e, Cliente c
WHERE dataInicio <= e.DataEntrega
    AND dataFim >= e.DataEntrega
    AND e.Cliente_NIF = c.NIF;
```

Figura 16 - Procedure EncomendasNumIntervaloDatas

O utilizador em causa fornece as datas de início e final do intervalo de tempo em que pretende listar as encomendas e estas são apresentadas com o número da encomenda, a data do pedido, a data em que esta foi entregue e nome do cliente ao qual esta se destina.

Se um utilizador inserir na pesquisa uma data de início posterior à data de fim, a pesquisa não ocorre.

5.3.3 Listar os produtos que são dirigidos para um Cliente

Esta query é utilizada quando um gestor empresarial pretende listar os produtos das encomendas que são dirigidas para um cliente em específico.

```
CREATE PROCEDURE ProdutosDasEncomendasParaUmCliente (IN nomeClienteX varchar(50))
SELECT p.nome, p.preco, pe.quantidade
FROM Produto p, Produto_Encomenda pe, Encomenda e, Cliente c
WHERE p.idProduto = pe.idProduto
    AND pe.NrEncomenda = e.NrEncomenda
    AND e.Cliente_NIF = c.NIF
    AND c.nome=nomeClienteX;
```

Figura 17 - Procedure ProdutosDasEncomendasParaUmCliente

O gestor fornece o nome do cliente que pretende saber as informações relacionadas. A lista é devolvida com o nome e o preço do produto, assim como a quantidade que foi dirigida para o cliente requerido.

5.3.4 Informar sobre os créditos numa determinada data

Esta query é utilizada quando um gestor empresarial pretende saber qual o crédito obtido numa determinada data.

```
CREATE PROCEDURE CreditosNumaData (IN dataX DATE)
SELECT e.DataPedido, sum(p.preco * pe.quantidade)
FROM Encomenda e, Produto p, Produto_Encomenda pe
WHERE p.idProduto = pe.idProduto
      AND pe.NrEncomenda = e.NrEncomenda
      AND dataX = e.DataPedido
GROUP BY e.DataPedido;
```

Figura 18 - Procedure CreditosNumaData

O gestor fornece a data em que pretende obter a informação dos créditos da empresa. Os dados são devolvidos com a data em questão e o valor dos créditos totais, que é obtido através da soma do valor de venda de todos os produtos das encomendas desse dia.

5.4. Tradução das transações estabelecidas para SQL

Uma *transaction* é uma operação lógica atómica que muda o estado duma base de dados. Esta pode ser inicializada por um utilizador ou por um programa através de, por exemplo, um SELECT, um INSERT, ou um UPDATE na base de dados. As mudanças que ocorrem durante uma transação não são visíveis até esta estar concluída. Estas não são necessariamente concluídas.

5.4.1 Permitir ao funcionário de escritório adicionar um novo cliente

```
DELIMITER $$
CREATE PROCEDURE AdicionarNovoCliente(IN id INT, n VARCHAR(50), r VARCHAR(50), c VARCHAR(40), e VARCHAR(40), t INT)
BEGIN
    DECLARE erro BOOL DEFAULT 0;
    DECLARE existe BOOL DEFAULT 0;
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET erro = 1;

    START TRANSACTION;
    IF ((SELECT NIF FROM Cliente WHERE NIF = id)>0) THEN SET existe = 1;
    END IF;

    IF existe
        THEN SELECT('O Cliente já existe na Base de Dados') AS msg;
        SET erro = 1;
    ELSE
        INSERT
            INTO Cliente(NIF, nome, rua, cidade, endereco, tipo)
            VALUES (id, n, r, c, e, t);
        SET erro = 0;
    END IF;

    IF erro THEN
        SELECT ('Erro ao adicionar uma entrada na tabela Cliente') AS msg;
        ROLLBACK;
    ELSE
        SELECT ('Operação efetuada com sucesso') AS msg;
        COMMIT;
    END IF;
END;
```

Figura 19 - Transaction AdicionarNovoCliente

Esta transação tem como objetivo inserir um cliente. Caso o cliente já exista na base de dados, é realizado um *rollback* de forma a abortar a transação e voltar ao ponto antes do início da transação. Caso não ocorra nenhum erro, é realizado um *commit* e são realizadas as alterações pretendidas à base de dados, neste caso a inserção de um cliente.

5.4.2 Permitir ao motorista adicionar data de entrega e a distância da encomenda

```

DELIMITER $$

CREATE PROCEDURE AdicionarDataEntrega(IN nE INT, d FLOAT)
BEGIN
    DECLARE erro BOOL DEFAULT 0;
    DECLARE nulo BOOL DEFAULT 0;
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET erro = 1;

    START TRANSACTION;

    IF ((SELECT e.dataEntrega FROM Encomenda e WHERE e.NrEncomenda=nE) IS NULL) THEN SET nulo = 1;
    END IF;

    IF nulo
        THEN UPDATE Encomenda SET dataEntrega = NOW() WHERE nrEncomenda = nE;
        UPDATE Encomenda SET distancia = d WHERE nrEncomenda = nE;
        ELSE SET erro = 1;
    END IF;

    IF erro THEN
        SELECT ('Esta Encomenda já foi entregue') as msg;
        ROLLBACK;
    ELSE
        SELECT ('Operação efetuada com sucesso') as msg;
        COMMIT;
    END IF;
END; $$
```

Figura 20 - Transaction AdicionarDataEntrega

Esta transação tem como objetivo adicionar a data de entrega e a distância duma encomenda. O *rollback* na transação é realizado se a encomenda já foi entregue, e por consequência, já possui uma data de entrega. Caso este campo se encontre a *null*, significa que a encomenda ainda não foi entregue, logo é possível atualizar o valor para a data em que o utilizador se encontra, sendo este o processo de *commit*. Importante referir que estas ações são realizadas pelos motoristas da empresa, pelo que o utilizador neste caso é um motorista.

5.5. Escolha, definição e caracterização de índices em SQL

Para efetuar a consulta de tabelas numa base de dados, nomeadamente se estas tiverem muitos registo, os índices são um recurso que torna essa consulta muito mais eficiente e otimizada. Porém, esta utilização não deve ser abusiva para não tornar esta utilização numa penalidade devido à grande quantidade de memória que vai ser gasta.

O MySQL, automaticamente, atribui índices às chaves primárias das nossas tabelas, ou seja, todas estas chaves pertencentes a tabelas serão índices.

Para além dos índices automaticamente atribuídos às chaves primárias, decidimos também adicionar alguns índices a atributos que achamos necessário. É o caso do nome do produto, na entidade Produto, pois não existem produtos iguais, sendo assim possível encontrar um produto através do seu nome. O atributo Designacao, da entidade Tipo_Produto, também foi colocado como índice para encontrar um Tipo de um Produto. A mesma razão permitiu colocar o atributo Designacao, da entidade Tipo_Cliente, como índice. Por último, como todos os endereços de clientes são diferentes, escolhemos o atributo Endereco, da entidade Cliente, como índice para a procura de cliente.

5.6. Estimativa do espaço em disco da base de dados e taxa de crescimento anual

Para estimar o espaço necessário em disco, precisamos de calcular o espaço ocupado por cada relação, em número de bytes. Assim, começamos por isso:

Relação Veículo Transportador

Atributo	Tipo	Espaço ocupado(bytes)
Matrícula	VARCHAR (10)	10
marca	VARCHAR (15)	15
NrKmsRealizados	INT	4
anoFabrico	INT	4
dataInspecao	DATE	8
dataRenovacaoSeguro	DATE	8
dataRenovacaoImpostoSelo	DATE	8
Condutor	VARCHAR (40)	40
NrArmazem	INT	4
Total por entrada	--	101

Relação Armazem

Atributo	Tipo	Espaço ocupado(bytes)
NrArmazem	INT	4
NrCamioes	INT	4
Rua	VARCHAR (50)	50
Cidade	VARCHAR (40)	40
Total por entrada	--	98

Relação Armazem_Produto

Atributo	Tipo	Espaço ocupado (bytes)
NrArmazem	INT	4
idProduto	INT	4
Stock	INT	4
Total por entrada	--	12

Relação Produto

Atributo	Tipo	Espaço ocupado(bytes)
idProduto	INT	4
Nome	VARCHAR (50)	50
Preco	FLOAT	4
Tipo	INT	4
Total por entrada	--	62

Relação Tipo_Produto

Atributo	Tipo	Espaço ocupado (bytes)
idTipoProduto	INT	4
Designação	VARCHAR (15)	15
Total por entrada	--	19

Relação Produto_Encomenda

Atributo	Tipo	Espaço ocupado (bytes)
NrEncomenda	INT	4
idProduto	INT	4
Quantidade	INT	4
Total por entrada	--	12

Relação Encomenda

Atributo	Tipo	Espaço ocupado(bytes)
NrEncomenda	INT	4
dataPedido	DATE	8
dataEntrega	DATE	8

Distancia	FLOAT	4
VeiculoTransportador_matricula	VARCHAR (10)	10
Cliente_NIF	INT	4
Total por entrada	--	38

Relação Cliente

Atributo	Tipo	Espaço ocupado(bytes)
NIF	INT	4
Nome	VARCHAR (50)	50
Rua	VARCHAR (50)	50
Cidade	VARCHAR (40)	40
Endereço	VARCHAR (40)	40
Cliente_NIF	INT	4
Total por entrada	--	188

Relação Contacto

Atributo	Tipo	Espaço ocupado (bytes)
Contacto	VARCHAR (45)	45
Cliente_NIF	INT	4
Total por entrada	--	49

Relação Tipo_Cliente

Atributo	Tipo	Espaço ocupado (bytes)
IdTipoCliente	INT	4
Designacao	VARCHAR (15)	15
Total por entrada	--	19

Necessitamos agora de estabelecer um povoamento tipo para poder estimar o espaço TOTAL ocupado.

Esse povoamento será, então: 3 armazéns, que possuem um total de 55 produtos, que são distribuídos para 18 clientes. Existem 5 tipos de produtos e 8 tipos de clientes. Os 18 clientes possuem, cada um 1 contacto, ou seja, 18 contactos. A frota da empresa é de 5 veículos. A empresa possui 42 encomendas até à data. Os 55 produtos encontram-se nos 3 armazéns logo $55 * 3 = 165$ Armazem_Produto. Como existem 42 encomendas, também existem 42 relações Produto_Encomenda.

Para este tipo de povoamento, o espaço ocupado seria aproximadamente:

$$3 * 98 + 55 * 62 + 18 * 188 + 5 * 19 + 8 * 19 + 18 * 49 + 5 * 101 + 42 * 38 + 165 * 12 + 42 * 12 = 12\ 802 \text{ bytes} = 0.012802 \text{ megabytes.}$$

5.6.1 Taxa de crescimento anual

Para determinar a taxa de crescimento anual, vamos supor que diariamente se registam 5 clientes, todos eles possuindo contacto. Os armazéns recebem todas as semanas um NOVO produto. Todos os anos a frota é expandida em 1 veículo. Todos os dias são feitas 10 encomendas.

$$365 * (5 * 188 + 5 * 49 + 10 * 38) + 52 * 19 + 101 = 572\ 314 \text{ bytes} = 0.572314 \text{ MB}$$

5.7. Definição e caracterização das vistas de utilização em SQL

Nesta secção apresentamos três das vistas de utilização que implementámos no nosso sistema. Uma vez que as vistas de utilização são definidas como sendo tabelas que contêm dados depois de ser aplicada uma determinada operação, achamos por bem definir algumas vistas de utilização com o intuito de ficar com informação armazenada numa tabela à parte e operar sobre essa tabela.

De seguida apresentamos então, três das vistas implementadas.

5.7.1 Filtrar os Veículos Transportadores que necessitam de ser substituídos

Esta view serve para listar todos os Veículos Transportadores que já atingiram uns certos quilómetros de utilização e necessitam de ser substituídos por outros.

```

CREATE VIEW VeiculosNecessitamSubstituicao AS
SELECT v.matricula, v.marca, v.dataInspecao
FROM VeiculoTransportador v
WHERE v.NrKmsRealizados >= 300000;

```

Figura 21 - Vista VeiculosNecessitamSubstituicao

Esta vista funciona conjuntamente com uma *trigger*, que quando uma encomenda é entregue, os quilómetros são adicionados ao camião e esta vista é atualizada com o intuito de saber se algum veículo atingiu os 300 mil quilómetros.

5.7.2 Filtrar os Veículos Transportadores que necessitam de renovar o imposto de selo

Esta *view* serve para listar todos os Veículos Transportadores que necessitam de renovar o imposto de selo no mês corrente.

```

CREATE VIEW VeiculosRenovarImpostoSelo AS
SELECT v.matricula, v.marca, v.dataInspecao
FROM VeiculoTransportador v
WHERE MONTH(v.dataRenovacaoImpostoSelo) = MONTH(NOW())
    AND YEAR(v.dataRenovacaoImpostoSelo) = YEAR(NOW());

```

Figura 22 - Vista VeiculosRenovarImpostoSelo

Atua em conjunto com uma *trigger* que aquando duma troca de mês atualiza esta view.

5.7.3 Listar Encomendas pendentes

Esta *view* serve apenas para listar todas as encomendas que não têm data de entrega, e consequentemente assumidas como pendentes.

```

CREATE VIEW EncomendasPendentes AS
SELECT *
FROM Encomenda e
WHERE e.dataEntrega IS NULL
ORDER BY e.dataPedido;

```

Figura 23 - Vista EncomendasPendentes

5.8. Definição e caracterização dos mecanismos de segurança em SQL

Um sistema de base de dados tem de assegurar dois tipos de segurança, a do sistema e a dos dados. No nosso caso de estudo, a segurança do sistema é assegurada tanto pelo sistema de base dados, como pelas aplicações que foram referidas nos capítulos introdutórios (mobile e desktop). Da parte das aplicações espera-se que filtrem cada utilizador e lhe atribuam um tipo. Da parte do sistema de base de dados criámos 4 papéis, como já foi referido na secção 2.3. Desta forma o acesso à base de dados pode ser feito apenas por estes 4 tipos de utilizadores.

De forma a garantir a segurança de dados é necessário restringir as ações de cada tipo de utilizador no sistema de base de dados. Seria absurdo um chefe de armazém adicionar um novo cliente ou marcar uma encomenda, tal como seria estranho que fosse um gestor a adicionar um novo produto e associá-lo a um armazém. As ações que cada tipo de utilizador pode realizar na base de dados foram descritas levemente na secção 2.3 e nos requisitos de controlo.

Realçamos então a criação de tipos de utilizadores e alguns tipos de permissões atribuídos a um tipo de utilizadores.

```
CREATE ROLE Motorista;
```

```
CREATE ROLE FuncionarioEscritorio;
```

```
CREATE ROLE Gestor;
```

```
CREATE ROLE ChefeArmazem;
```

Figura 24 - Criação de Utilizadores

```
GRANT EXECUTE  
ON PROCEDURE VeiculosRenovarSeguroMesCorrente  
TO FuncionarioEscritorio;
```

Figura 25 - Permissão para um Funcionário de Escritório

```
GRANT EXECUTE  
ON PROCEDURE CreditosNumaData  
TO Gestor;
```

Figura 26 - Permissão para um Gestor

5.9. Revisão do sistema implementado com o utilizador

Ao longo deste capítulo fomos analisando todos os detalhes de construção do modelo físico, especificando as tabelas geradas, as transações realizadas, as vistas, as diferentes *procedures*, detalhes de segurança, etc. Verificamos então, que o modelo físico cumpre todos os requisitos anteriormente estipulados, tal foi confirmado com a posterior validação por parte do utilizador que aceitou o modelo como uma representação “real” do seu negócio.

6. Conclusões e Trabalho Futuro

Realizamos este projeto com o intuito de aplicar os conhecimentos adquiridos sobre Bases de Dados Relacionais. Produzimos então, uma solução para um problema que havia surgido numa empresa de distribuições, que não conseguia gerir o seu negócio devido ao armazenamento desordenado e ineficiente dos dados, mas sobretudo à quantidade abrupta de dados que continuava a aumentar face ao crescimento da empresa. A solução encontrada foi a construção de uma base de dados, funcionando em conjunto com duas aplicações, de forma a providenciar uma plataforma de dados com qualidade, organização, consistência e com a devida correção.

Nesta base de dados criámos uma ligação entre Encomendas, Produtos, Clientes, Veículos Transportadores e Armazéns que deu como resultado uma organização de informação relativa à DistribuMinho que facilita, por completo, o acesso a dados de encomendas que foram tratadas pela empresa. Desde questões básicas como obter informações relativas a qualquer uma destas entidades, até a questões mais complexas como a verificação da necessidade de fazer a inspeção de um veículo num determinado mês.

Importante também mencionar que decidimos criar uma relação entre Armazém e Veículo Transportador de forma a otimizar a velocidade das entregas, por exemplo, se a encomenda for feita na zona de Viana do Castelo, apenas utilizar os veículos que estão alocados no Armazém de Viana do Castelo.

No que diz respeito às funcionalidades do SGBD podemos identificar algumas, sendo estas: as interrogações dos utilizadores, as transações e as vistas de utilização.

Nas interrogações dos utilizadores foram apresentados vários casos, entre eles “Permitir a consulta das informações dum cliente” e “Informar sobre os créditos numa determinada data”. Tanto uma como a outra são bastante importantes, a primeira para analisar dados referentes a mercado (uma *query* mais simples), e a segunda para analisar dados referentes a gestão financeira da empresa (uma *query* mais complexa), o que demonstra a versatilidade das *queries* realizadas pelo grupo.

Nas transações analisamos um caso de inserção e um caso de atualização. Ambos representam funcionalidades necessárias para o bom funcionamento da empresa e, em geral, demonstram o objetivo destas transações criadas para a base de dados.

As vistas de utilização analisadas no relatório assinalam bem o intuito destas, isto porque representam informação que decidimos guardar em tabelas específicas para ser mais o acesso a dados específicos que são, normalmente, mais procurados. No caso de o utilizador necessitar de observar as encomendas que ainda estão por ser entregues, a *view* explicada no capítulo 5.7.3 permite que seja simples esta ação.

Finalizando o nosso projeto, o grupo sente que o principal objetivo foi cumprido e as necessidades inerentes a este foram satisfeitas. Consideramos que, com este sistema de base de dados, a empresa em questão tornar-se-ia mais funcional e permitiria que esta se focasse mais noutras setores.

7. Sistema de Base de Dados Não Relacional

O termo NoSQL (*Not Only SQL*) descreve soluções de armazenamento de base de dados não relacionais. Pode assim dizer-se que uma base de dados NoSQL, além de não ter interface SQL, é open source e completamente distinta do modelo relacional tradicional, isto porque as bases de dados NoSQL foram justamente projetadas a partir de necessidades que as bases de dados tradicionais relacionais não satisfaziam, como **alta performance** e **capacidade de expansão**. Podemos afirmar que existem quatro tipos diferentes de base de dados NoSQL, que são os seguintes:

- Chave-Valor: Armazena os seus dados num padrão chave-valor, fazendo lembrar as tabelas de hash. Ex: MemcacheD, Riak, REDIS.
- **Grafos**: Armazena os seus dados na forma de grafo, com vértices e arestas. Ex: Neo4j, Sesame.
- Colunas: Armazena os seus dados em linhas particulares de uma tabela no disco. Ex: Cassandra, Hibase;
- Documento: armazena os seus dados como documentos, onde um documento pode ser um dado num formato chave-valor (por exemplo o padrão JSON). Ex: MongoDB, CouchDB.

7.1. Justificação da utilização de um sistema NoSQL

7.1.1 Armazenamento de Dados

- SQL

O armazenamento é num modelo relacional, em tabelas, em que as linhas são as entradas/entidades e as colunas os atributos das respetivas entidades.

- NoSQL

O NoSQL abrange uma série de base de dados, cada uma com a sua estrutura de armazenamento de dados.

7.1.2 Estrutura e Flexibilidade

- SQL

Alterações que sejam feitas a nível de estrutura implicam alterar toda a base de dados.

- NoSQL

Estruturas dinâmicas. Informações podem ser adicionadas facilmente e permite haver entradas em colunas da tabela vazias.

7.1.3 Escalabilidade

- SQL

A escalabilidade é vertical. Assim, mais dados implica haver um servidor maior o que poderá ser dispendioso. É possível escalar uma base de dados em vários servidores, mas isso é um processo difícil e demorado.

- NoSQL

A escalabilidade é horizontal, como já referido anteriormente. Mais económico.

7.1.4 Propriedades ACID

- SQL

A grande maioria dos bancos de dados relacionais são compatíveis com ACID.

- NoSQL

Varia de acordo com as tecnologias, mas muitas soluções NoSQL sacrificam a compatibilidade ACID para desempenho e escalabilidade.

7.1.5 Vantagens

- Dados sempre disponíveis;
- Custo mais reduzido que uma base de dados relacional;
- Base de dados orientada a objetos flexíveis;
- Facilidade em introduzir novos dados;
- Excelente maneira de lidar com o problema de dados em massa.

7.2. Identificação e descrição dos objetivos da base de dados, em termos de aplicações e de utilizadores

As bases de dados relacionais não permitem que os dados sejam representados através de grafos de uma forma natural, com isso, algumas pesquisas tornam-se extremamente complexas, ou até mesmo inconcebíveis nas bases de dados relacionais. Uma base de dados baseada em grafos utiliza o modelo de grafos para representar o esquema. Esse modelo apresenta três componentes básicas: nodos (vértices), os relacionamentos entre nodos (arestas) e as propriedades dos nodos ou dos relacionamentos existentes entre eles. Assim, nas bases de dados baseadas em grafos basta apenas navegarmos pela estrutura de dados desse grafo para conseguirmos obter os resultados pretendidos.

Desta forma, os principais objetivos da base de dados são a intenção de responder com o maior volume de dados da forma mais rápida possível, a qualquer tipo de query e, ainda assim, garantir toda a veracidade dos dados.

7.3. Questões que serão realizadas sobre o SBD NoSQL

Nesta segunda fase do trabalho definimos os requisitos tendo em conta os requisitos já definidos na fase SQL. Ainda que o modelo de implementação esteja ligeiramente alterado, as necessidades serão as mesmas e o sistema terá que responder a todas estas.

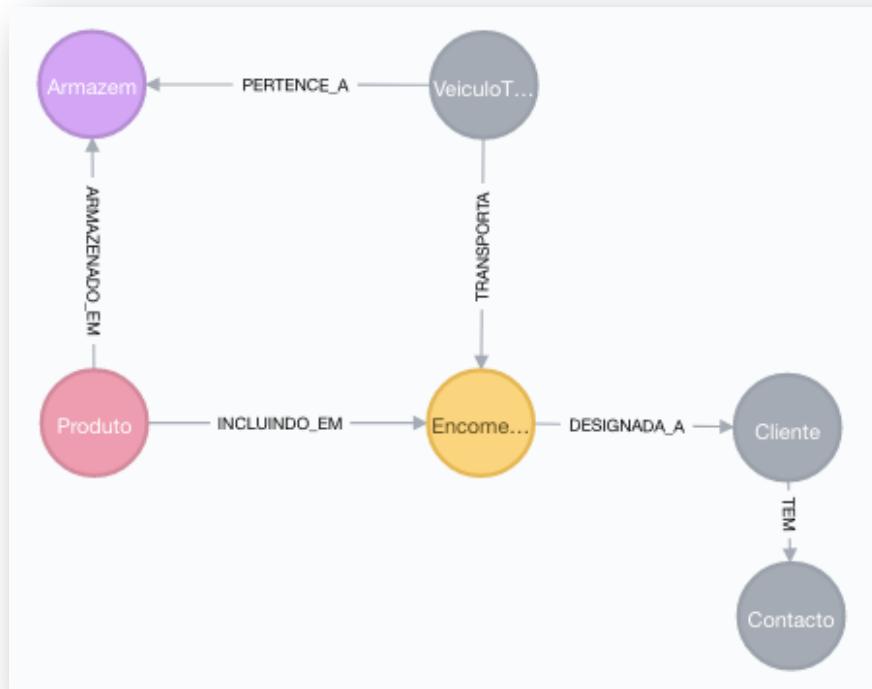
Posto isto, são apresentadas algumas questões que serão abordadas nesta fase do projeto:

- 1) Permitir a consulta das informações dum Produto.
- 2) Listar as Encomendas que decorreram num intervalo de datas.
- 3) Listar as Encomendas que têm um determinado produto.
- 4) Filtrar os Veículos Transportadores que necessitam de ser substituídos, no mês corrente.

Esquema de Base de Dados

Como nas bases de dados não relacionais mantém-se a informação de como os nós são relacionados entre si, é desnecessário a criação de chaves estrangeiras, de tabelas com relacionamento “N:M”, tabelas para atributos multi-valorados e outras tabelas como, por exemplo, tabelas de tipos de forma a ter melhor qualidade de armazenamento de dados.

Logo foi desenvolvido o seguinte esquema:



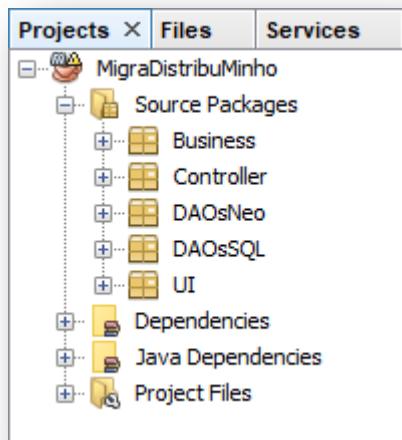
Identificamos de seguida os atributos (com valor de exemplo) de cada nó:

- **Cliente**
 - NIF : “123764376”
 - Nome : “Mercado Acácia”
 - Rua : “Rua Terroeiras”
 - Cidade : “Vila Real”
 - Endereço : “acaciaOnTheStreet@gmail.com”
 - Tipo : “MiniMercado”
- **Armazém**
 - NrArmazem : “2”
 - NrCamioes : “2”
 - Rua : “Rua Poço da Prata”
 - Cidade : “Braga”
- **Produto**
 - idProduto : “3”
 - Nome : “Fiambre Peito Frango – Nobre”
 - Preço : “11.99”
 - Tipo : “Charcutaria”

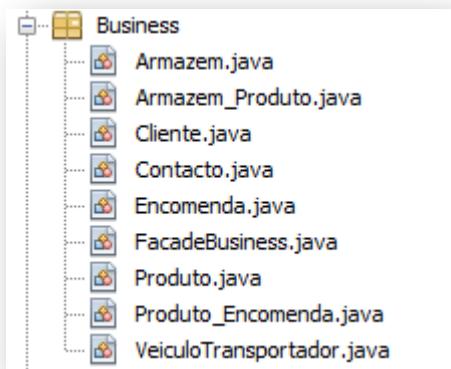
- **Veículo Transportador**
 - Matricula : “20-VR-86”
 - Marca : “Volvo”
 - Nr Kms Realizados : “40000”
 - Ano Fabrico : “2017”
 - Data de Inspeção : “2018/11/30”
 - Kms Máximos : “30”
 - Data Renovação Seguro : “2019/11/04”
 - Data Renovação Imposto de Selo : “2019/11/04”
 - Condutor : “José António”
- **Encomenda**
 - Nr Encomenda : “1”
 - Data de pedido : “2018/11/04”
 - Data de entrega : “2018/11/05”
 - Distância : “92.9”
- **Contacto**
 - Contacto 1 : “259672019”
 - Contacto 2 : “123764376”

7.4. Objetos de Dados para Alimentar SBD NoSQL

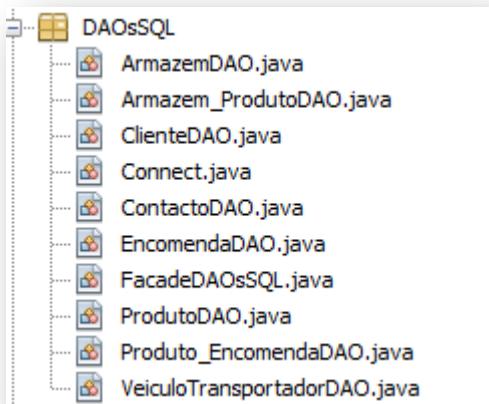
Para alimentar o sistema foi criada uma aplicação através do programa - *NetBeans*.



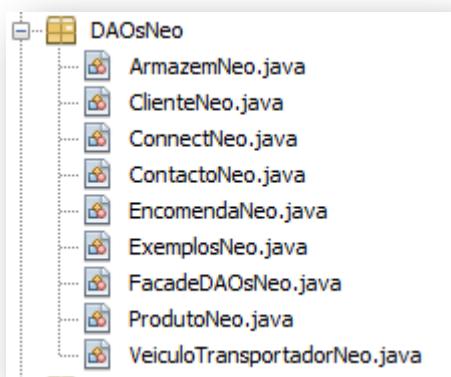
Como ilustra a figura, foram criadas cinco *packages* : *Business*, *Controller*, *DAOsNeo*, *DAOsSQL* e *UI*. De seguida explicaremos as funcionalidades de cada uma.



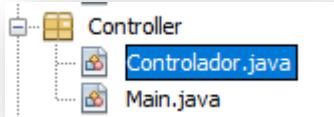
No *Business* encontram-se definidas as classes que servirão para a criação de novos objetos quando estes são retirados da base de dados definida em SQL para serem passados para Neo4j.



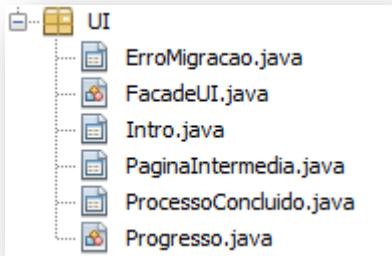
No package DAOsSQL é onde são feitos os métodos para retirar os valores de cada uma das tabelas necessárias da base de dados relacional, SQL.



No package DAOsNeo é onde são feitos os métodos para colocar os valores retirados na base de dados não relacional, Neo4j.



No Controller encontram-se os métodos de exportação e importação dos dados na sua totalidade.



No package UI está definida a interface da aplicação.

7.5. Processo de Migração

Nesta secção será descrito todo o processo que foi realizado para migrar os dados dum contexto relacional para um contexto não relacional. Para tal desenvolvemos um pequeno programa em Java que realizasse toda esta tarefa.

Este processo foi dividido em 3 etapas, que são respetivamente, extração, transformação e carregamento.

7.5.1 Extração de Dados

Nesta primeira etapa baseamo-nos no modelo lógico da base de dados relacional, criando uma classe para cada relação existente, excetuando apenas as relações *Tipo_Produto* e *Tipo_Cliente*. Estas classes tem o intuito de aceder à base de dados e retirar os dados que residem em cada tabela. Utilizamos os conectores oferecidos pelo MySQL para realizar tal operação. Um exemplo pode ser visto na figura seguinte onde se extraem todos os dados relativos a um **Cliente**.

```

public class ClienteDAO {
    public List<Cliente> getClientes() throws SQLException, Exception{
        Connection c = Connect.connect();
        List<Cliente> resultado = new ArrayList<>();
        if(c!=null) {
            PreparedStatement ps = c.prepareStatement("SELECT * FROM Cliente;");
            ResultSet rs = ps.executeQuery();
            while(rs.next()) {
                Cliente res = new Cliente();
                res.setNif(rs.getInt("NIF"));
                res.setNome(rs.getString("nome"));
                res.setRua(rs.getString("rua"));
                res.setCidade(rs.getString("cidade"));
                res.setEndereco(rs.getString("endereco"));

                PreparedStatement ps1 = c.prepareStatement("SELECT Designacao FROM Tipo_Cliente WHERE idTipoCliente = ?;");
                ps1.setInt(1,res.getNif());
                ResultSet rs1 = ps1.executeQuery();
                if(rs1.next())
                    res.setDesignacao(rs.getString("Designacao"));
                else res.setDesignacao(null);

                resultado.add(res);
            }
        } else{
            throw new Exception("Unable to establish connection");
        }
        Connect.close(c);
        return resultado;
    }
}

```

Figura 27 - Médo de extração de dados dos Clientes

7.5.2 Transformação de Dados

As transformações dos dados são efetuadas nas classes faladas na subsecção anterior. Estas classes tanto fazem extração como transformação dos dados.

Aquando da extração, os dados são colocados em variáveis de instância de objetos que correspondem a cada tabela da base de dados relacional. Criamos objetos cujas variáveis de instância armazenam os atributos de cada relação do sistema. Exemplificando, para a relação *Cliente* temos uma classe *ClienteDAO* que extrai os dados da base de dados e transforma em instâncias do objeto *Cliente*.

```

public class Cliente {
    int nif;
    String nome;
    String rua;
    String cidade;
    String endereco;
    String designacao;
}

```

Figura 28 - Classe Cliente

7.5.3 Carregamento de Dados

Nesta última fase, procedemos então à inserção dos dados que residem na aplicação como objetos, num ambiente não relacional, mais propriamente numa base de dados orientada a grafos suportada pelo *Neo4j*. Para tal, seguindo a mesma lógica temos uma classe

encarregue de fazer o carregamento de cada tipo de objetos. Continuando com o exemplo do **Cliente**, existe então uma classe *ClienteNeo*, que para todos os objetos do tipo Cliente cria um nó no grafo com as propriedades deste nó a serem as variáveis de instância do objeto *Cliente*.

Os acessos ao *Neo4j* são feitos através dos drivers por este fornecidos. Para a inserção de qualquer tipo de objeto é realizada uma transação, para que caso existam erros, estes não sejam propagados para a base de dados não relacional.

A seguinte imagem pode ajudar na compreensão da ideia desenvolvida.

```
public class ClienteNeo {
    public void migraClientes(List<Cliente> clientes){
        Driver driver = ConnectNeo.connection();
        Session session = driver.session();

        try (Transaction tx = session.beginTransaction()){
            for(Cliente c : clientes){
                tx.run("CREATE (:Cliente{nif:{nif}, nome:{n}, rua:{r}, cidade:{c}, endereco:{e}, designacao:{d}})",
                      parameters("nif", c.getNif(), "n", c.getNome(), "r", c.getRua(), "c", c.getCidade(),
                                 "e", c.getEndereco(), "d", c.getDesignacao()));
            }
            tx.success();
        }
        ConnectNeo.close(driver);
    }
}
```

Figura 29 - Método de Carregamento de Clientes

7.6. Detalhes da Extração dos Dados

7.6.1 Criação dos Nós

- **Cliente**

```
public void migraClientes(List<Cliente> clientes){
    Driver driver = ConnectNeo.connection();
    Session session = driver.session();

    try (Transaction tx = session.beginTransaction()){
        for(Cliente c : clientes){
            tx.run("CREATE (:Cliente{nif:{nif}, nome:{n}, rua:{r}, cidade:{c}, endereco:{e}, designacao:{d}})",
                  parameters("nif", c.getNif(), "n", c.getNome(), "r", c.getRua(), "c", c.getCidade(),
                             "e", c.getEndereco(), "d", c.getDesignacao()));
        }
        tx.success();
    }
    ConnectNeo.close(driver);
}
```

Figura 30 - Código de Criação de Nós Cliente

- **Produto**

```

public void migraProdutos(List<Produto> produtos){
    Driver driver = ConnectNeo.connection();
    Session session = driver.session();

    try (Transaction tx = session.beginTransaction()){
        for(Produto p : produtos){
            tx.run("CREATE (a:Produto {idProduto:{na},nome:{nc},preco:{r},Designacao:{c}})",
                  parameters("na", p.getIdProduto(),"nc",p.getNome(),"r",p.getPreco(),
                             "c",p.getDesignacaoTipo()));
        }
        tx.success();
    }
    ConnectNeo.close(driver);
}

```

Figura 31 - Código de Criação de Nós do Tipo Produto

- **Encomenda**

```

public void migraEncomendas(List<Encomenda> encomendas){
    Driver driver = ConnectNeo.connection();
    Session session = driver.session();

    try (Transaction tx = session.beginTransaction()){
        for(Encomenda c : encomendas){

            String dp = String.valueOf(c.getDataPedido());
            String de = String.valueOf(c.getDataEntrega());

            tx.run("CREATE (a:Encomenda{nrEncomenda:{nrenc},dataPedido:{dp},dataEntrega:{de},distancia:{d},",
                   + "veiculoTransportador_matricula:{vt},cliente_nif:{cnif}})",
                   parameters("nrenc", c.getNrEncomenda(),"dp",dp,"de",de,"d",c.getDistancia(),
                              "vt",c.getVeiculoTransportador_matricula(), "cnif", c.getCliente_nif()));

        }
        tx.success();
    }
    ConnectNeo.close(driver);
}

```

Figura 32 - Código de Criação de Nós do Tipo Encomenda

- **Armazém**

```

public void migraArmazens(List<Armazem> armazens){
    Driver driver = ConnectNeo.connection();
    Session session = driver.session();

    try (Transaction tx = session.beginTransaction()){
        for(Armazem ar : armazens){
            tx.run("CREATE (a:Armazem {NrArmazem:{na},NrCamioes:{nc},rua:{r},cidade:{c}})",
                  parameters("na", ar.getNrArmazem(),"nc",ar.getNrCamioes(),"r",ar.getRua(),"c",ar.getCidade()));
        }
        tx.success();
    }
    ConnectNeo.close(driver);
}

```

Figura 33 - Código de Criação de nós do Tipo Armazém

- **Veículo Transportador**

```

public void migraVeiculosTransportadores(List<VeiculoTransportador> veiculos){
    Driver driver = ConnectNeo.connection();
    Session session = driver.session();

    try (Transaction tx = session.beginTransaction()){
        for(VeiculoTransportador v : veiculos){

            String dinsp = String.valueOf(v.getDataInspecao());
            String rs = String.valueOf(v.getRenovacaoSeguro());
            String ris = String.valueOf(v.getRenovacaoImpostoSelo());

            tx.run("CREATE (a:VeiculoTransportador{matricula:{m},marca:{mar},nrKmsRealizados:{nrkms},"
                + "anoFabrico:{af},dataInspecao:{di},kmsMaximos:{kmax},renovacaoSeguro:{rs},"
                + "renovacaoImpostoSelo:{ris},condutor:{con},nrArmazem:{nrar}})",
                parameters("m", v.getMatricula(), "mar", v.getMarca(), "nrkms", v.getNrKmsRealizados(),
                    "af", v.getAnoFabrico(), "di", dinsp, "kmax", v.getKmsMaximos(), "rs", rs, "ris",
                    "ris", "con", v.getCondutor(), "nrar", v.getNrArmazem()));
        }
        tx.success();
    }
    ConnectNeo.close(driver);
}

```

Figura 34 - Código de Criação de Nós do Tipo Veículo Transportador

7.6.2 Criação dos Relacionamentos entre Nós

- Relacionamento Veículo Transportador **PERTENCE A** Armazém

```

public void migraVeiculosTransportadores_Armazens(List<VeiculoTransportador> veiculos){
    Driver driver = ConnectNeo.connection();
    Session session = driver.session();

    try (Transaction tx = session.beginTransaction()){
        for(VeiculoTransportador v : veiculos){
            tx.run("MATCH (v:VeiculoTransportador) WHERE v.matricula = {matricula}" +
                "MATCH (ar:Armazem) WHERE ar.NrArmazem = {arm}" +
                "CREATE (v) -[:PERTENCE_A]-(ar)", parameters("matricula", v.getMatricula(),
                    "arm", v.getNrArmazem()));
        }
        tx.success();
    }
    ConnectNeo.close(driver);
}

```

Figura 35 - Criação de Relacionamentos :PERTENCE_A

- Relacionamento Produto **ARMAZENADO EM** Armazém

```

public void migraArmazens_Produtos(List<Armazem_Produto> ar_produtos){
    Driver driver = ConnectNeo.connection();
    Session session = driver.session();

    try (Transaction tx = session.beginTransaction()){
        for(Armazem_Produto ap : ar_produtos){

            tx.run("MATCH (p:Produto{idProduto:{id}})" +
                "MATCH (ar:Armazem{NrArmazem:{arm}})" +
                "CREATE (p) -[:ARMAZENADO_EM{stock:{sto}}]-(ar)",
                parameters("id", ap.getIdProduto(), "arm", ap.getNrArmazem(), "sto", ap.getStock()));
        }
        tx.success();
    }
    ConnectNeo.close(driver);
}

```

Figura 36 - Criação de Relacionamentos :ARMAZENADO_EM

- Relacionamento Produto **INCLUÍDO EM** Encomenda

```

public void migraProdutos_Encomendas(List<Produto_Encomenda> produtos_enc){
    Driver driver = ConnectNeo.connection();
    Session session = driver.session();

    try (Transaction tx = session.beginTransaction()){
        for(Produto_Encomenda pe : produtos_enc){

            tx.run("MATCH (p:Produto{id:{id}})" +
                  "MATCH (e:Encomenda{nrEncomenda:{nre}})" +
                  "CREATE (p) -[:INCLUIDO_EM{quantidade:{q}}]-> (e)",
                  parameters("id", pe.getIdProduto(), "nre", pe.getNrEncomenda(), "q", pe.getQuantidade()));
        }
        tx.success();
    }
    ConnectNeo.close(driver);
}

```

Figura 37 - Criação de Relacionamentos :INCLUIDO_EM

- Relacionamento Encomenda **DESIGNADA A** Cliente

```

public void migraEncomendas_Clientes(List<Encomenda> encomendas){
    Driver driver = ConnectNeo.connection();
    Session session = driver.session();

    try (Transaction tx = session.beginTransaction()){
        for(Encomenda e : encomendas){
            tx.run("MATCH (e:Encomenda{nrEncomenda:{nre}})" +
                  "MATCH (c:Cliente{nif:{nif}})" +
                  "CREATE (e) -[:DESIGNADA_A]-> (c)",
                  parameters("nre", e.getNrEncomenda(), "nif", e.getCliente_nif()));
        }
        tx.success();
    }
    ConnectNeo.close(driver);
}

```

Figura 38 - Criação de Relacionamentos :DESIGNADA_A

- Relacionamento Veículo Transportador **TRANSPORTA** Encomenda

```

public void migraEncomendas_Veiculos(List<Encomenda> encomendas){
    Driver driver = ConnectNeo.connection();
    Session session = driver.session();

    try (Transaction tx = session.beginTransaction()){
        for(Encomenda e : encomendas){
            tx.run("MATCH (e:Encomenda{nrEncomenda:{nre}})" +
                  "MATCH (v:VeiculoTransportador{matricula:{matricula}})" +
                  "CREATE (v) -[:TRANSPORTA]-> (e)",
                  parameters("nre", e.getNrEncomenda(), "matricula", e.getVeiculoTransportador_matricula()));
        }
        tx.success();
    }
    ConnectNeo.close(driver);
}

```

Figura 39 - Criação de Relacionamentos :TRANSPORTA

7.7. Queries em Neo4j

7.7.1 Permitir a consulta das informações dum Cliente

```
MATCH (c:Cliente {NIF:"5"})
RETURN c
```

Figura 40 - Querie Neo4j

7.7.2 Listar as Encomendas que decorreram num intervalo de datas

```
MATCH (e:Encomenda)
WHERE (split(e.dataPedido, " "))[0] >= "2018-11-04" AND (split(e.dataEntrega, " "))[0] <= "2018-11-12"
RETURN e
```

Figura 42 - Querie Neo4j

7.7.3 Listar os produtos que são dirigidos para um Cliente

```
MATCH (p:Produto)-[:INCLUINDO_EM]->(e:Encomenda)-[:DESIGNADA_A]->(c:Cliente {Nome:"Ricardo"})
RETURN p,e,c
```

Figura 44 - Querie Neo4j

7.8. Conclusão

Após a realização deste trabalho o grupo sente ter adquirido as competências necessárias para trabalhar com uma base de dados *NoSQL* e considera agora entender melhor as principais diferenças entre uma base de dados relacional e uma base de dados não relacional.

Nesta parte do projeto apenas foi necessário adaptar os modelos anteriormente elaborados através de várias etapas, ajudando-nos a perceber todos os passos necessários que permitem a transformação de uma base de dados relacional a uma não relacional.

O uso do Neo4j permitiu perceber outra perspetiva sobre a criação de uma base de dados, pois este modelo é constituído por grafos e nós.

A adaptação a outra linguagem de programação para criar nós, relacionamentos e realizar queries também foi um ponto muito positivo na realização deste projeto.

O grupo confirma tudo o que nos foi transmitido nas aulas sobre as bases de dados *NoSQL* e o modelo Neo4j e acha que a principal vantagem da utilização deste modelo é o elevado grau de distribuição dos dados, permitindo mais solicitações aos dados do que um SBD relacional.

Dando por concluído o projeto, achamos que a realização do projeto no seu todo (modelo relacional e não relacional) permitiu-nos perceber como ambos os modelos funcionam e têm a sua importância no mercado de trabalho.

Referências

- Connolly, T., Begg, C., Database Systems, A Practical Approach to Design, Implementation, and Management, Addison-Wesley, 4a Edição, 2004.
- Ian Robinson, Jim Webber, Emil Eifrem, Graph Databases, New Opportunities for Connected Data, O'Reilly Media, 2^a Edição, Junho de 2015

Lista de Siglas e Acrónimos

SGBD Sistema de Gestão de Base de Dados

