UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS

Nilmar Luiz Guarda Junior

Desenvolvimento de algoritmo para sistema de gestão de distribuição de estoques

Nilmar Luiz Guarda Junior

Desenvolvimento de algoritmo para sistema de gestão de distribuição de estoques

Relatório submetido à Universidade Federal de Santa Catarina como requisito para a aprovação na disciplina **DAS 5511: Projeto de Fim de Curso** do curso de Graduação em Engenharia de Controle e Automação.
Orientador: Prof. Leandro Buss Becker

Florianópolis
2019

Nilmar Luiz Guarda Junior

Desenvolvimento de algoritmo para sistema de gestão de distribuição de estoques

Esta monografia foi julgada no contexto da disciplina DAS5511: Projeto de Fim de Curso e aprovada na sua forma final pelo Curso de Engenharia de Controle e Automação.

Florianópolis, 08 de fevereiro de 2019

Banca Examinadora:

Igor Gois
Orientador na Empresa
Bix Tecnologia

Prof. Leandro Buss Becker
Orientador no Curso
Universidade Federal de Santa Catarina

Prof. João Carlos Espíndola Ferreira
Avaliador
Universidade Federal de Santa Catarina

Luiz Arthur D'Ávila da Silva Prazeres
Debatedor
Universidade Federal de Santa Catarina

Ricardo Ventura
Debatedor
Universidade Federal de Santa Catarina

AGRADECIMENTOS

Agradeço à Universidade Federal de Santa Catarina, em especial ao Departamento de Automação e Sistemas, pelos esforços e suporte necessário para a formação de cidadãos.

Aos professores, pela paciência, disponibilidade e conhecimentos compartilhados. Em especial, ao orientador Professor Leandro Becker, pela confiança e apoio no desenvolvimento desta monografia.

Ao orientador Engenheiro Igor Gois, pela confiança e apoio no desenvolvimento deste projeto.

Agradeço à empresa Bix Tecnologia, pelo conhecimento disponibilizado, confiança e incentivo ao desenvolvimento profissional.

Aos colegas de estágio, pelo conhecimento compartilhado e companhia no dia a dia.

Agradeço à minha família, pelo suporte, amor, compreensão, e incondicional apoio.

À minha companheira Fernanda Bassoli, pela paciência, motivação e incentivo.

E a todos que partilharam dos momentos de minha caminhada acadêmica.

RESUMO

Recentemente, com a aplicação de novas ferramentas computacionais e demais tecnologias de software, as instituições comercias têm apresentado mudanças no modo de gerir seus diferentes setores. Com estas aplicações, serviços e estratégias tradicionalmente executadas podem ser realizadas com maiores velocidades e de modo antes impraticáveis. Tais avanços devem compreender todas as atividades ligadas ao fluxo de transformação que o produto passa até estar disponível para o consumidor final, de modo que a instituição possa monitorar e reduzir seus custos, a fim de perdurar na atual concorrência do mercado. Neste cenário, o projeto aqui documentado descreve o processo de concepção de algoritmos para cálculos de distribuição de estoque e da comunicação dos resultados para uma página *web*. Tais aplicações compõem uma ferramenta para auxílio na tomada de decisão do funcionário deste setor de uma empresa com mais de uma centena de filiais em todo território nacional. Para isso, são levantadas as teorias envolvidas no projeto, seguido das ferramentas utilizadas para sua concepção. Na sequência, tem-se a explanação do desenvolvimento das aplicações e dos resultados obtidos com este trabalho.

Palavras-chave: Distribuição de estoque. Data Integration. ETL. API.

ABSTRACT

Lately, with the application of new computational tools and other softwares, commercial enterprises have submitted changes in the way to handle their differents sectors. With this applications, services and strategies traditionally performed can be applyed faster and in a way previously unviable. Such advances should cover all the activities linked to the stream of product transformation until be avaible to the final consumer, so that the institution can track and reduce costs to stand in the current market competition. In this scenario, the project hereby documented describes the process of conception of algorithms to calculate the distribution of stock and the communication of the results to a web page. These applications are parts of a tool for assist the sector employee decision of a enterprise with more than a hundred of subsidiaries in national territory. For this purpose, teories involved to the project are approached, followed of the used tools for its conception. Subsequently, is exposed an explanation of application developed and the results achieved with this project.

Key-words: Distribution of stocks. Data Integration. ETL. API.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

API - *Application Programming Interface*

CSV - *Comma-separated values*

DDL - *Data Definition Language*

ERP - *Enterprise Resource Planning*

ETL - *Extration Transform Load*

HTML - *Hypertext Markup Language*

HTTP - *HyperText Transfer Protocol*

I/O - *Input/Output*

JDBC - *Java Database Conectivity*

JSON - *JavaScript Object Notation*

KPI - *Key Performance Indicator*

OLAP - *On-Line Analytical Processing*

OLTP - *Online Transaction Processing*

OSI - *Open System Interconnection*

PDI - *Pentaho Data Integrator*

RDBMS - *Relational Database Management Systems*

REST - *Representational State Transfer*

SGDB - Sistemas de Gestão de Base de Dados

SQL - *Structered Query Language*

TCP - *Transmission Control Protocol*

UML - *Unified Modeling Language*

URL - *Uniform Resource Locator*

XML - *Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Recentemente as indústrias e as instituições comerciais têm apresentado mudanças significativas, o que influencia no modo com que se administra todo e qualquer setor destas organizações. Tais mudanças criam a necessidade de melhorias e de atualizações para concorrer no mercado sem abrir mão de sua receita (GARCIA, MACHADO, *et al.*, 2006).

Sendo os principais motivos desta mutação o aumento e a facilidade do fluxo de informações através da internet e das tecnologias informacionais em toda a cadeia comercial, as instituições são coagidas a utilizar destes novos componentes para atingir os objetivos impostos e manter-se neste novo mercado (TURBAN, RAINER e POTTER, 2003).

Neste cenário, as ferramentas computacionais e demais tecnologias de software e de hardware tem influência direta nestes avanços, fazendo com que serviços e estratégias, antes tradicionalmente executadas, sejam praticadas com maiores velocidades e de modo antes inacessíveis (GARCIA, MACHADO, *et al.*, 2006). Além disso, para perdurar na concorrência da atual conjuntura econômica, monitorando e reduzindo constantemente os custos, o emprego destas inovações deve compreender todas as atividades ligadas ao fluxo de transformação que o produto passa até estar disponível ao consumidor final (CÉSARO, 2007). O decrescente custo da tecnologia juntamente com a facilidade de seu uso incentiva os analistas utilizarem deste novo meio de coleta, armazenamento, processamento e visualização de dados com maior eficiência e rapidez (NAZÁRIO, 2011).

Indiscutivelmente, o estoque e a logística são setores cruciais dentro de qualquer organização, pois determinam diretamente o abastecimento de matérias para seus clientes, sendo fatores decisivos no sucesso ou fracasso de uma unidade comercial (CÉSARO, 2007). Atualmente, níveis de estoque direcionados à logística, na grande maioria das empresas, são determinados de maneira empírica, sem fazer uso de cálculos automatizados e precisos, contradizendo a tendência de oportunidade de redução de custos em qualquer setor da organização. Dessa maneira, os gestores não encontram justificativas

convincentes para definir quantidades destinadas à distribuição ou estocagem, oportunizando custos adicionais dentro destas áreas.

Assim como nos demais setores, é de suma importância a aplicação de tecnologias que auxiliem os tomadores de decisões a encontrar informações que realmente refletem o cenário, ou seja, informações confiáveis e atualizadas. Como resultado, existe uma significativa melhoria na eficiência da distribuição de produtos que possibilita, além da melhoria do serviço, reduções de custos que justificam os valores investidos (NAZÁRIO, 2011).

A empresa Bix Tecnologia atua no setor de consultoria e comércio de soluções de informática com o intuito de tornar a gestão das empresas-clientes mais eficiente. Tendo como principais frentes o *Business Intelligence* e *Data Analytics,* seu meio de atuação circunda inúmeros outros temas abordados na área de tecnologia de informação, ciência de dados e automação gerencial. A empresa, tendo expertise em todos esses conceitos, oferece a seus clientes produtos específicos para suas necessidades, independente do nicho de atuação os quais estão imersos e do setor dentro da organização.

O projeto aqui documentado fora encomendado para a Bix Tecnologia por um cliente que atua no ramo de vestuário, que, em busca de melhorias em seus setores, notou a necessidade de aprimorar o processo de tomada de decisão do analista de estoque e logística, na definição de quais produtos direcionar do centro de distribuição para cada unidade de venda, bem como a quantidade de cada um.

1.1 Problemática

O cliente, que possui seis marcas de roupas comercializadas em mais de cento e setenta franquias físicas em todo o Brasil, além do comercio on-line, é considerado um dos maiores grupos de moda do país. Fundada em 2010 com a fusão de duas de suas marcas, a empresa apresenta grande prestigio no comercio de vestimentas com uma receita anual na casa de um bilhão de reais.

Não diferente da maioria das grandes empresas, o grupo está em busca de constantes melhorias, investindo fortemente no ramo da tecnologia com a criação de um setor focado em soluções digitais.

Neste contexto, no setor de estoque e logística, viu-se a necessidade de aprimorar o processo do analista de estoque na tomada de decisão sobre quantidades e quais produtos direcionar do centro de distribuição para cada unidade de venda.

Para suprir a demanda de produtos nas unidades de venda, o centro de distribuição direciona para o setor de logística certas quantidades dos diferentes produtos disponíveis em estoque. Estas proporções nem sempre condizem com o que é necessário em cada franquia, o que gera custos extras de transporte e de estocagem para a empresa, refletindo diretamente no custo de oportunidade de capital.

Além disso, os principais problemas consequentes de uma distribuição equivocada as unidades de varejo são conhecidos como *overstock* e *stock out.* O primeiro, é caracterizado pela quantidade a mais do que o necessário numa unidade de venda, resultando em perda do capital de giro, uma vez que o capital se encontra imobilizado, além de resultar em custo de armazenamento e a possibilidade de obsolescência do produto estagnado. Já o *stock out,* tem consequências ainda piores. Caracterizado pela falta de produto em estoque, tem como principais implicações o impacto sobre a marca e até a perda de clientes. Ambos os casos resultam no lucro não realizado.

O atual sistema, além de pouco automatizado, demanda elevado tempo de espera de seu usuário, uma vez que necessita de downloads de bases atualizadas de uma central e de armazenamento destes em diretórios específicos. Este procedimento, além de abrir brechas para falhas, torna-se trabalhoso para o operador, como na alteração de dados que resultam na falta de fidelidade dos valores atingidos. Por se tratar de uma ferramenta de pouca agilidade para a função de extração e seleção dos dados do banco, imprime no processo um tempo de operação na casa de horas.

1.2 Objetivos

Como objetivo, a empresa busca mais agilidade no processo de cálculo de distribuição do estoque, com dados mais confiáveis numa plataforma mais robusta de operação.

1.2.1 Objetivo Geral

Tem-se como principal objetivo tornar ágil o processo de decisão de distribuição de peças de vestuário dos centros de estoque, com resultados fidedignos à realidade. Isto se dará através do projeto e do desenvolvimento de uma ferramenta com foco no auxílio ao profissional responsável da área. Para isso, tem-se a automação dos cálculos que resultam na ordenação deste montante, provenientes dos dados da empresa, e a disposição destes resultados a seus usuários, fazendo uso de ferramentas apropriadas para o desenvolvimento e execução de tais funções.

O algoritmo terá função de resultar dados confiáveis de maneira mais rápida que o atual processo, através da busca no banco de dados pelos elementos necessários para os cálculos com o uso de algoritmos em ferramentas próprias para manipulação de dados, tornando o processo mais ágil e livre de falhas provenientes de interferências indevidas.

Com isso, as informações encaminhadas ao setor de logística auxiliam o usuário a definir de forma embasada e precisa as quantidades a serem ordenadas para cada filial.

1.2.2 Objetivos Específicos

Este projeto tem como objetivos específicos interpretar as regras de distribuição do atual sistema utilizado na empresa cliente e, assim, utiliza-las para o desenvolvimento de algoritmos que fomentarão a interface da ferramenta de distribuição hospedada em um site da web, além de desenvolver a comunicação entre estas diferentes partes do sistema. Os algoritmos terão como funcionalidade a extração e o processamento dos dados do banco da empresa, cálculos matemáticos envolvendo estes dados, e a comunicação entre o algoritmo e a interface da ferramenta hospedada na web.

Seguem os objetivos de forma detalhada:

- Interpretar e documentar as atuais regras de distribuição do sistema em uso;
- Extrair informações necessárias do banco de dados;

- Automatizar a execução de manipulações e cálculos com estes dados;
- Comunicar-se com a interface informacional;
- Memorizar as informações utilizadas e recebidas da interface;
- Registrar os resultados das distribuições;
- Validar a ferramenta em uso.

1.3 Metodologia

A metodologia para o desenvolvimento deste projeto seguiu o conceito da *Action-Research* (pesquisa-ação). Este modelo, segundo (ENGEL, 2000), é considerado diferente da pesquisa tradicional, pois busca unir a pesquisa à ação prática. Desse modo, também se enquadra num método onde o autor tem afinidade com a prática e busca aumentar sua compreensão desta.

Adotando-se a metodologia exposta, o desenvolvimento do projeto segue as seguintes etapas:

I. Tem-se um problema para resolver;

Apresentado o cenário pelo cliente, busca-se estudar a problemática visando embasar o desenvolvimento futuro.

As principais adversidades do processo utilizado pelo cliente são: demora na definição da distribuição de estoques; processo operoso; e a falta de fidelidade nos resultados obtidos.

Com isso, tem-se a necessidade de desenvolver uma ferramenta que vise atender as funcionalidades por este requisitada.

II. Projeta-se um artefato que se acredita solucionar o problema;

Estudado a problemática, análises são feitas em busca de encontrar as melhores ferramentas para sua solução considerando a expertise de atuação da empresa Bix Tecnologia. As possibilidades levantadas são analisadas em conjunto com a equipe de suporte da empresa cliente e a melhor alternativa é selecionada.

Com isso, projeta-se a ferramenta em sua completude composta de diferentes partes que juntas e em comunicação visam, principalmente, diminuir

ao máximo a demanda do usuário final na definição da distribuição, que trate os dados de maneira ágil e confiável e informe com praticidade os resultados ao usuário.

III. Implementar tal artefato;

Definido o projeto da ferramenta, para a implementação do algoritmo que automatizará os cálculos e as lógicas de distribuição do estoque, utiliza-se a ferramenta *open source* Pentaho, que trata com maestria grandes volumes de dados provenientes de diferentes fontes.

A interface com usuário, onde é efetuada a entrada de informações da distribuição, é realizada por uma página web já existente. Assim, para interligar as duas aplicações, uma API será desenvolvida, a qual é requisitada pela página e receberá os dados necessários para a distribuição. Estes, devem ser inseridos num banco de dados que será acessado pelo algoritmo para execução dos cálculos.

IV. Realizam-se experimentos/testes sobre o artefato desenvolvido;

Desenvolvida a ferramenta, diferentes validações são realizadas para cada etapa do processo, avaliando todos os cálculos e todos os processos de comunicação.

V. Verifica-se se a solução atende os objetivos;

Por fim, através das análises realizadas na etapa anteriormente descrita, é verificado junto ao cliente a funcionalidade da ferramenta, e se esta atinge os resultados desejados. Caso resultado não seja convincente, levantam-se hipóteses do que pode ser realizado para que se alcance os resultados pretendidos.

1.4 Estrutura do trabalho

Este documento conta com 6 capítulos, estruturados na seguinte forma:

Inicialmente, o capítulo 1 conta com a apresentação do tema, da problemática que se pretende solucionar, e dos objetivos gerais e específicos do projeto, e metodologia, além da própria estrutura do documento.

O capítulo 2 contém o referencial teórico e tecnológico, abordando em cada subitem os conteúdos presentes no projeto, como banco de dados, ETL

(*Extract Transfer Load*), modelagem UML (*Unified Modeling Language*) e Web API.

No capítulo 3 é tratado o projeto e especificações, contando com a apresentação dos requisitos funcionais e não-funcionais, além de diagramas de modelagem UML

O capítulo 4 descreve a implementação do sistema aqui descrito, iniciando com a apresentação das ferramentas utilizadas seguido dos processos realizados para a concepção do sistema, separados em interpretação dos cálculos, desenvolvimento do algoritmo e desenvolvimento da WebAPI.

Em seguida, o capítulo 5 aborda os resultados obtidos com o desenvolvimento deste projeto, contando com análises de seus benefícios e amostras de sua interface.

Por fim, o capítulo 6 apresenta as considerações finais e sugestões de trabalhos futuros.

2 REFERENCIAL TEÓRICO E TECNOLÓGICO

Neste capítulo são apresentadas as teorias e tecnologias envolvidas no desenvolvimento deste projeto. Inicialmente, tem-se a explanação sobre banco de dados e suas funcionalidades. Em seguida, é abordado o tema ETL. Por fim, trata-se do conteúdo relacionada a comunicação da ferramenta, denominada WebAPI. Estas teorias serão utilizadas na prática, apresentadas no capítulo subsequente.

2.1 Banco de dados

A coleção de dados armazenados em determinados esquemas e tabelas é conhecido como banco de dados (SILBERSCHATZ , KORTH e SUDARSHAN, 2006). A interação e a manipulação dos usuários com os dados têm como propósito gerar informações relevantes sobre estes.

Informações são parte do capital de qualquer empresa, tal como são suas cadeiras, mesas e escritórios. Sendo assim, como um bem, deve ser utilizada de maneira estratégica para que se possa tanto traçar como atingir objetivos e metas (RODRIGUES MACHADO e ABREU, 2009).

Recentemente, com o advento e popularidade da internet, aumentou-se significativamente a utilização dos bancos de dados e de suas informações. Inúmeros serviços, antes realizados pessoalmente, foram convertidos em aplicações Web com acesso a bancos.

Junto destes avanços, sistemas de gerenciamento de banco de dados (SGBD) e softwares para manipulação de dados, juntamente com modelos e arquiteturas de banco surgiram para dar suporte no gerenciamento desta gama de informações (SILBERSCHATZ , KORTH e SUDARSHAN, 2006).

2.1.1 Modelo Relacional

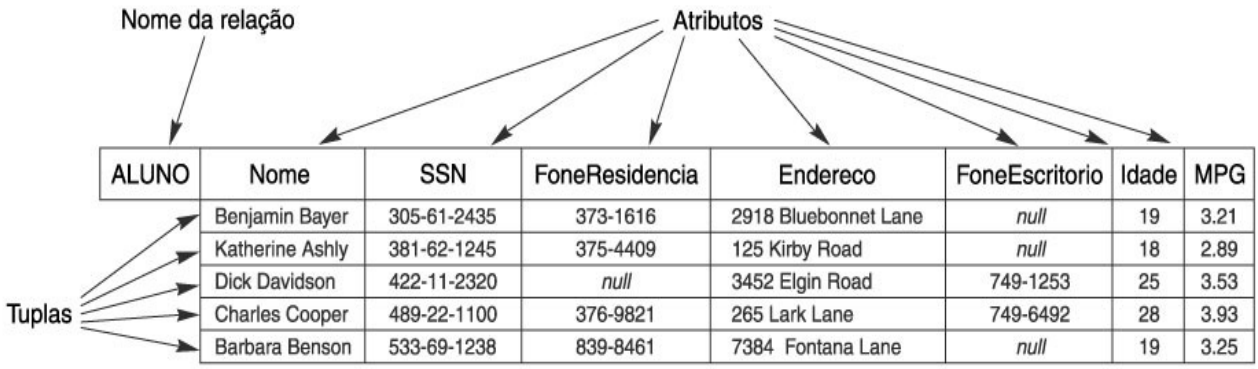
Conforme citado, os bancos de dados seguem determinados modelos, que nada mais são do que conceitos que descrevem os dados, relações de

dados, semânticas e restrições (SILBERSCHATZ , KORTH e SUDARSHAN, 2006).

Um dos modelos mais utilizado é o relacional. Neste, os dados são organizados em relações. Cada relação pode ser vista como uma tabela, contendo colunas e linhas. Cada coluna corresponde a atributos e cada linha à tuplas ou elementos da relação. Comumente, estes são chamados de registros e aqueles de campos (FRANCO, 2013).

Esta visão de dados oferece um conceito simples e familiar de estruturação de dados, o que explica o sucesso deste modelo. Na Figura 1, tem-se um exemplo desta estrutura.

Figura 1 - Os atributos e as tuplas de uma relação ALUNO.



ALUNO	Nome	SSN	FoneResidencia	Endereco	FoneEscritorio	Idade	MPG
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	*null*	19	3.21
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	*null*	18	2.89
	Dick Davidson	422-11-2320	*null*	3452 Elgin Road	749-1253	25	3.53
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	*null*	19	3.25

Fonte: SILBERSCHATZ , KORTH e SUDARSHAN (2006).

2.1.2 Chaves de Ligação

Outro conceito importante num banco de dados relacional são os atributos chave ou chaves de ligação. Com o seu uso, pode-se criar ligações entre diferentes tabelas de um banco, tendo assim uma relação entre estas.

Comumente, por boa prática, opta-se por separar os atributos numa tabela conforme seu assunto principal. Por exemplo, ao mapear os dados relativos a uma livraria, teremos diferentes relações como livros, editoras, autores, venda, etc. Ao armazenar as informações relativas a cada relação, separa-se os atributos de cada relação citada, sem repetir e misturar os dados.

Por exemplo, para a tabela “Livros”, tem-se os atributos relativos somente aos livros; para a relação “Autores”, informações dos autores, e assim por diante.

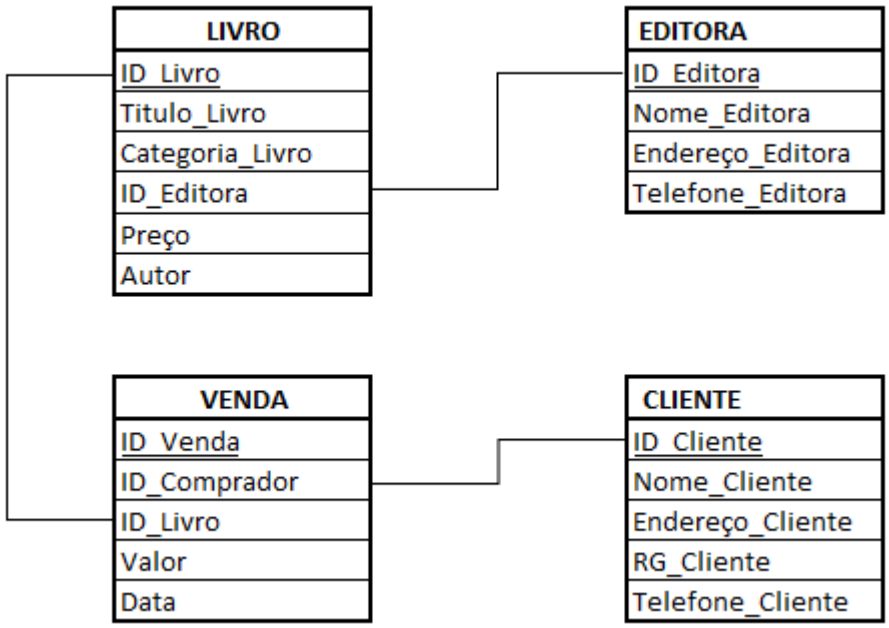
Considera-se no cenário citado, a adição de um novo exemplar no acervo da livraria. Tal informação, quando adicionada ao banco, acarretaria no acréscimo de dados em todas as tabelas citadas. Assim, para relacionar estes dados referentes ao mesmo livro, utiliza-se os atributos chaves.

Considerando que cada tupla contém uma combinação de atributos única, podemos identificar cada uma através de um único atributo, desde que apresente um valor diferente para cada tupla (FRANCO, 2013).

Na falta de um atributo com tal característica, são criados atributos com um código de identificação, geralmente nomeados incialmente por ID. Trazendo este conceito para o exemplo citado, diferentes exemplares podem apresentar um mesmo título na livraria. Neste caso, o atributo Título não satisfaria a condição de unicidade requerida. Uma alternativa, então, é a criação de uma coluna ID para cada exemplar adicionado à livraria, com este servindo de chave.

Para a ligação com as demais tabelas ser funcional, a mesma coluna chave de uma tabela deve estar presente na tabela que se deseja correlacionar, sem necessariamente ter o mesmo nome. Por exemplo, para a tabela “Vendas”, além de seus próprios atributos como “Data” e “Preço”, teria também a coluna chave do exemplar vendido. O exemplo é ilustrado abaixo na Figura 2.

Figura 2 - Relação entre tabelas.



Fonte: O Autor.

Com este modelo implementado, pode-se, então, descobrir os demais dados dos exemplares envolvidos em certa venda interligando os dados das tabelas.

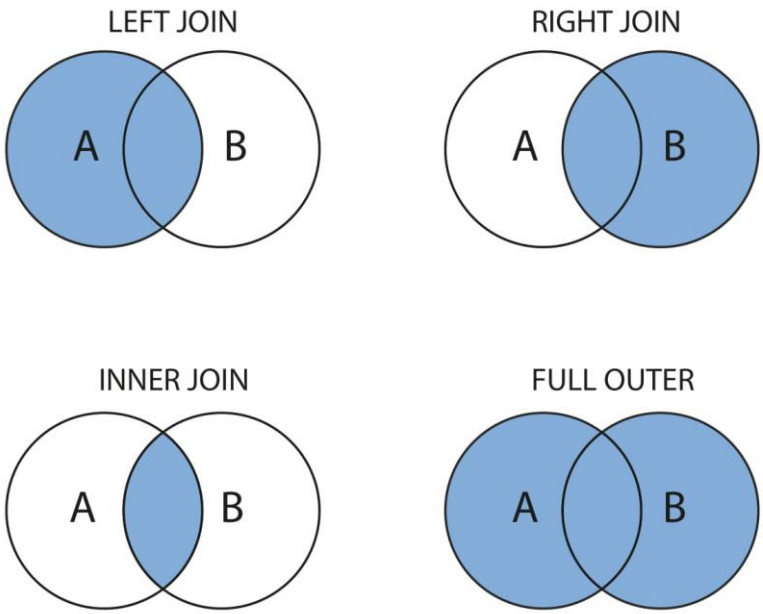
As vantagens de trabalhar com chaves de ligação vai além do fator organizacional de um banco de dados. Este modelo acelera o processo de acesso e consulta aos dados.

O uso de chaves de ligação está presente em distintas arquiteturas, cada uma visando requisitos específicos como agilidade em consultas, economia de armazenamento, entre outros. Um exemplo comum é o modelo *Star Schema*, presente em aplicações de *business intelligence*, na qual faz-se presente o uso de chaves na modelagem de suas tabelas.

A união de tabelas, ou *Join*, também é outro artifício muito utilizado que faz uso das chaves para sua execução. Neste processo, une-se atributos de diferentes tabelas em uma tabela pivô, centralizando os dados numa única tabela.

Existem diferentes modos de executar uma união, os principais são apresentados na figura abaixo. As esferas representam as tabelas e pode-se visualizar de maneira intuitiva a resultante do comando como a região preenchida, como na Figura 3.

Figura 3 - Principais tipos de *Joins*.



Fonte: O Autor.

2.1.3 SQL

A linguagem padrão para banco de dados relacionais é o SQL (*Structered Query Language*). Esta oferece uma interface declarativa de alto nível, facilitando seu entendimento ao usuário (SILBERSCHATZ , KORTH e SUDARSHAN, 2006). Apesar de ser utilizada em diversos SGDB relacionais, sua sintaxe é padronizada, divergindo em detalhes para cada produtora.

Os comandos podem tomar formas muito complexas, apesar disso, são baseados em clausulas simples.

Os principais comandos para definição de bancos são CREATE, ALTER e TRUNCATE. Quando mencionados, são acompanhados pelo objeto ao qual se referem, podendo ser as tabelas ou campos. Esta categoria de comandos também é nomeada DDL (*Data Definition Language*).

Os atributos são especificados quanto aos seus tipos, que podem ser dos mais variados. Os mais utilizados são numéricos, como *int,* e *big int,* e numéricos de ponto flutuante, *float* ou *real.* Também podem assumir cadeias de caracteres, declaradas *char,* quando possuem tamanho fixo, ou *varchar,* quando

variável. Outros tipos comumente usados são os booleanos, contendo valores *True* ou *False*; *Date* para datas; e *time stamp*, possuindo data e hora.

Quando se trata de consultas, ou *Querys,* como são comumente chamadas, os comandos fundamentais são: SELECT; FROM; e WHERE (opcional). Aquele, é seguido de quais atributos deseja-se ter como resposta da pesquisa; esse, especifica o nome de uma ou mais tabelas onde se encontra os atributos; e este, as restrições que os dados devem satisfazer para serem resultados da seleção. Tais comandos são atribuídos a categoria DML (*Data Manipulation Language*).

2.2 ETL

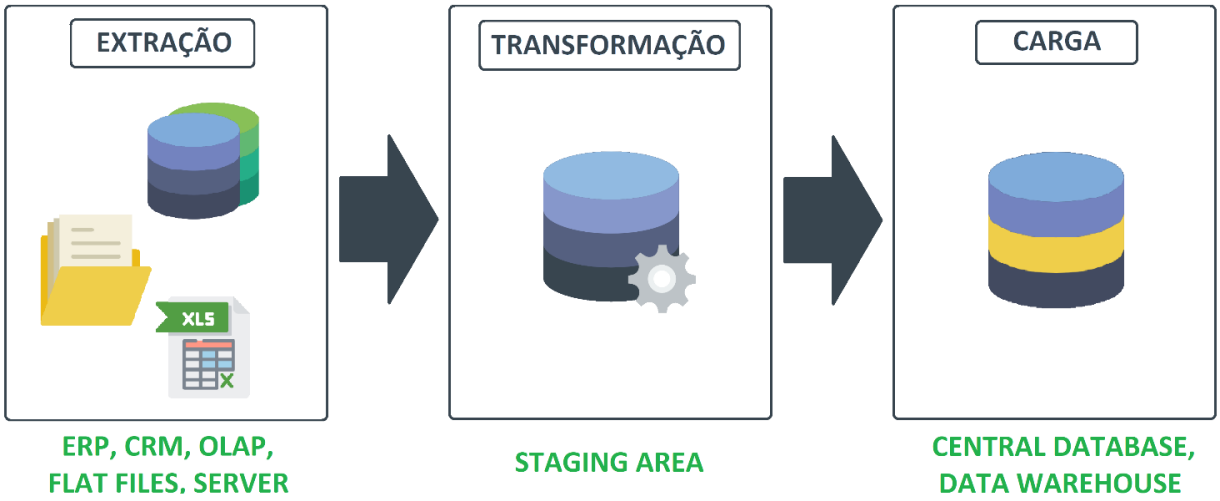
ETL é a sigla para o processo *Extract-Tranform-Load* ou Extração Transformação e Carga. Cada uma destas etapas guia os dados de sua origem, até sua apresentação. Entre estes extremos, a lógica é aplicada (KIMBALL e ROSS, 2013).

O processo de ETL é um dos componentes fundamentais para o sucesso de uma aplicação, pois trata diretamente com a integridade dos dados e com o desempenho final do mesmo. Assim, deve-se considerar o escopo do projeto geral ao implementar-se uma ETL, ponderando o volume de dados que terão de ser processados, juntamente com o tempo disponível para todo o processo, desde extração até a carga final.

Em muitos casos, o tema ETL é trazido associado com *business intelligence* e *Data Warehouses*. Isto se deve ao fato deste processo ser uma boa ponte entre dados brutos e informações baseadas em dados tratados.

O esquemático da Figura 4 representa um breve resumo do processo.

Figura 4 - Esquemático ETL.



Fonte: O Autor.

Os subcapítulos a seguir descrevem sequencialmente as etapas do processo.

2.2.1 Extração

Como primeira etapa do processo, os dados são lidos da fonte primária e conduzidos para uma *staging area*, de modo que o processo possa operar de forma independente do sistema de onde provem os dados.

Neste momento, é importante ter ciência do mapeamento das fontes dos dados necessárias para a aplicação, além dos tipos de dados que serão extraídos. Em boa parte dos casos, os dados são oriundos de diversos sistemas organizacionais (OLTP) do cliente.

A maioria dos produtos disponíveis voltados para ETL possibilitam a extração de diferentes formatos de fontes, como xml, xls, csv e SGDB, além de diferentes tipos de dados, como inteiros, decimais, datas, strings, entre outros (HENRY, HOON, *et al.*, 2005).

2.2.2 Transformação

Concluída a primeira etapa, os dados extraídos estão disponíveis para a aplicação da lógica desejada. Neste momento, os dados são manipulados conforme a necessidade da aplicação (KIMBALL e ROSS, 2013).

Como já citado, a arquitetura do modelo de dados também é um passo importante para a execução da ETL. Para isso, é na etapa de transformação que estes padrões são modelados através de uniões entre tabelas, exclusões de campos desnecessários e outras operações. Como exemplo de modelo, tem-se o *Star-Schema*, padrão que preza pelo desempenho largamente difundido em aplicações de *business intelligence.*

A sintaxe das tabelas extraídas nem sempre estão formatadas conforme esperado, dessa forma, os nomes dos atributos são editados para um melhor entendimento e manipulação, geralmente, utilizando caixa alta e poucos caracteres em seus cabeçalhos.

Nesta etapa, também, ocorrem a criação de campos resultantes de funções e rotinas para cálculos, como definições de KPIs (*Key Perfomance Indicators*), e indicadores de desempenho. A formatação dos dados também pode ser editada, como horário no formato de 12 para 24 horas, entre outros (HENRY, HOON, *et al.*, 2005).

2.2.3 Carga

A etapa final da ETL, a carga, é o processo onde os dados resultantes da etapa anterior são inseridos no banco de dados central. Estes fomentarão as aplicações com a informação já tratada e pronta para ser exibida na camada de visualização da aplicação.

O processo de carga pode variar para cada projeto, principalmente, no que se refere a temporização, reposição ou acréscimo de dados. Por exemplo, há casos em que a carga pode substituir os dados já existentes num determinado intervalo de tempo, pode adicionar a cada hora ou, ainda, adicionar dados mais recentes aos anteriormente carregados (processo conhecido como carga

incremental). Cada caso faz parte do escopo do projeto e depende da estratégia e usabilidade da aplicação final.

2.3 Web API

*Application Programing Interface (*Interface de Programação de Aplicações*)* ou API, é um conceito de aplicação que tem o objetivo de disponibilizar recursos para a utilização por outras diferentes aplicações, muitas vezes, restringindo acesso a estes e estabelecendo princípios específicos para seu uso (ORENSTEIN, 2000). Na maioria dos casos, seu uso não é evidenciado aos usuários da aplicação final.

As APIs são requisitadas pelos desenvolvedores através de chamadas dentro do algoritmo da aplicação principal com comandos específicos definidos pela aplicação solicitada. Como exemplo, programas que demandam algum serviço do sistema operacional provavelmente requisitam alguma API do próprio sistema.

Este tipo de aplicação é amplamente utilizado em serviços web, os quais são conhecidos como *Web Services*. Sua principal característica é o envio de mensagens através da rede, geralmente distinguidas entre *server-side* e *client-side*, com protocolo definido, diferentes arquiteturas e formatos de mensagens.

As mais utilizadas são WCF (*Windows Communication Foundation*) e, recentemente, REST (*Representational State Transfer*), fazendo uso do protocolo HTTP (*Hypertext Transfer Protocol)* com transferência de mensagens geralmente expressas em XML ou JSON. Estes temas serão abordados nos próximos subitens.

No caso deste projeto, tem-se a concepção de uma WebAPI, um tipo de *Web service* com padrão REST e troca de dados no formato JSON.

Como exemplo deste tipo de API, tem-se o Google Maps, onde outros sites utilizam seu serviço de mapas ou, ainda, gerenciadores de e-mails, que permitem a leitura e envio destas mensagens.

2.3.1 Protocolo HTTP

Hypertext Transfer Protocol, conhecido pela sigla HTTP, é um protocolo de transferência largamente utilizado na *World Wide Web.* Pertencente a camada de aplicação, segundo o Modelo OSI, tem como função descrever as mensagens que os clientes podem enviar ao servidor e quais respostas irão receber (TANENBAUM, 2002).

O modo habitual de conexão é o TCP (*Transmission Control Protocol*). Este, tem como vantagem principal sua confiabilidade, visto que trata de mensagens perdidas, duplicadas, longas ou confirmações em sua própria implementação, não exigindo nem do cliente nem do servidor tal compromisso (TANENBAUM, 2002).

Inicialmente desenvolvido para envios e respostas de um único item, ou seja, conexões não persistentes, este método era suficiente, uma vez que sua criação data de um período que páginas web eram baseadas em conteúdo de texto HTML. Como sabemos, o veloz advento da internet culminou em páginas com maiores dados, contendo imagens e mídias. Assim, uma numa nova versão deste protocolo, o HTTP 1.1 foi definido pela RFC 2616 (*Request for Comments*), a qual admite conexões persistentes, o que facilita a troca destes inúmeros dados.

No HTTP, tanto na requisição, quanto na resposta, as mensagens seguem padrões estruturais definidos no protocolo. Este, por sua vez, é constituído por uma linha inicial (denominada também como linha de requisição); por nenhuma ou por mais linhas de cabeçalho; por uma linha em branco para demarcar o fim do cabeçalho; e, por fim, o corpo da mensagem, podendo ser opcional dependendo da aplicação (KUROSE e ROSS, 2010).

A linha inicial é composta pelos campos método, URL e versão do HTTP. O primeiro, define o método da solicitação, em outras palavras, a ação sobre o recurso especificado no URL. São métodos internos ao protocolo: GET, POST, HEAD, PUT, DELETE, TRACE, CONNECT, OPTIONS.

O método GET é um dos mais utilizados e é empregado quando o cliente requisita um objeto descrito no campo do URL (KUROSE e ROSS, 2010). Neste tipo de método não há conteúdo no corpo da mensagem.

Já no método POST, o principal conteúdo está no corpo, ou *body.* A principal funcionalidade neste tipo de mensagem é a escrita de dados pelo cliente e o retorno do servidor atrelado a esta mensagem recebida. Um exemplo são páginas contendo vários campos a serem preenchidos e, posteriormente, submetidos pelo usuário.

Neste método, diferentes tipos de representações de informações podem ser transmitidos no corpo da mensagem. O tradicionalmente utilizado XML, vem dividindo espaço com o recente JSON (*JavaScript Object Notation*). Ambos descrevem dados serializados que facilitam a implementação das funcionalidades de sua tratativa pelo servidor.

Para cada solicitação numa comunicação HTTP, tem-se uma resposta que também segue o protocolo. Essa resposta é composta por uma linha de status e informações adicionais, que informam ao cliente o resultado de sua ação. Abaixo, é apresentado na tabela os cinco grupos de respostas, cada um com seu código, seguido do significado da resposta e de um exemplo.

Tabela 1 - Grupos de respostas.

Código	Significado	Exemplos
1xx	Informação	100 = server agrees to handle client's request
2xx	Sucesso	200 = request succeeded; 204 = no content present
3xx	Redirecionamento	301 = page moved; 304 = cached page still valid
4xx	Erro do cliente	403 = forbidden page; 404 = page can not found
5xx	Erro do servidor	500 = internal server error; 503 = try again later

Fonte: TANENBAUM, (2002).

2.4 Modelagem UML

A UML - *Unified Modeling Language,* ou Linguagem de Modelagem Unificada, é um padrão de linguagem visual que busca representar as perspectivas de um sistema de software através de diferentes artefatos gráficos (BEZERRA, 2015).

Tendo como principais contribuintes em seu desenvolvimento os pesquisadores Grady Booch, James Rumbaugh e Ivar Jacobson, a linguagem é baseada em padrões antes existentes e foi aprovada como padrão em meados

de 1997 (BEZERRA, 2015). Desde então, devido sua fácil interpretação ganhou muita popularidade, sendo muito utilizada no meio comercial e acadêmico.

São vários os diagramas que compõem o padrão UML, no entanto, cada um se adequa melhor a certas características presentes nos diversos tipos de sistemas de software. Para melhor descrever o projeto em questão, os diagramas concebidos serão apresentados no capítulo seguinte.

3 PROJETO E ESPECIFICAÇÕES

Em busca de solucionar os inúmeros impasses decorrentes do sistema de cálculo de distribuição de estoque utilizado pela empresa cliente, buscou-se modelar o problema e enquadrar a solução dentro das competências técnicas e de software cabíveis.

Em paralelo, a equipe da empresa cliente desenvolveu sua página web, que conta com os campos necessários para o usuário (analista de estoque e logística) acrescentar as informações necessárias para distribuição de estoque, entre eles, quantidades de peças, seus tamanhos e seus modelos, além das filiais que receberão a distribuição dos centros de estoque.

Essas informações serão comunicadas para o algoritmo pela página web, através da requisição da API desenvolvida, que terá como parte de suas funções transferir as informações da distribuição contidos na página para o banco de dados e, então, acionar o início da execução do algoritmo.

Para que sejam executados os cálculos com a informação relativa à distribuição correta, o identificador dos dados da distribuição inserida no banco é requisitado pela API no momento de sua inserção e repassado para o algoritmo junto de seu comando de inicialização.

O algoritmo é iniciado com a informação do identificador dos dados da distribuição a ser executada e começa o processo de cálculo das distribuições. Para isso ocorrer, processos de busca e união dos dados necessários provenientes de diferentes fontes, também conhecido como *Data Integration*, devem ser realizados de maneira otimizada, na busca de minimizar o tempo dispendido pelo funcionário nesta função.

Neste momento, devido a diferentes arquiteturas das fontes de dados utilizadas para os cálculos, deve-se utilizar manipulações e lógicas complexas, a fim de obter tabelas contendo todas as informações necessárias.

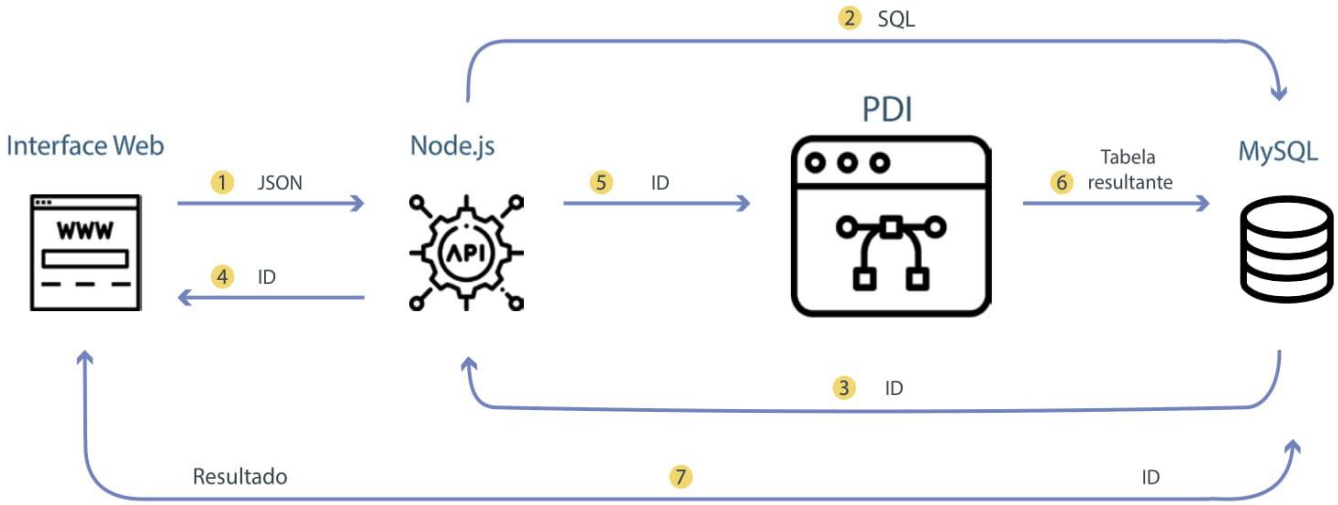
São inúmeros os atributos e as tuplas envolvidas nas operações da distribuição, visto que a quantidade de peças com seus diferentes atributos deve ser correlacionada com as filiais destino, e estas com seus históricos e velocidades de venda. Com isso, a massa de dados no fluxo do algoritmo torna-se grandiosa, além do resultado buscado acabar numa granularidade

especificamente pequena perante aos dados envolvidos até então. Assim, espera-se que as lógicas empregadas sejam otimizadas e as ferramentas utilizadas possam dar suporte a toda esta integração de informação.

Com a execução dos cálculos, a tabela de dados resultante é inserida no banco de dados central, contendo seu identificador. A página web, fazendo uso de outra API, resgata esses dados e os disponibiliza numa nova página informacional para seu usuário final.

O fluxo de informações até aqui descrito, é representado no esquemático da Figura 5. Nesta está representada cada aplicação desenvolvida que faz parte da ferramenta, com setas descrevendo o destino das mensagens trocadas e numerada com a ordem que seguem no fluxo.

Figura 5 - Fluxo de dados da ferramenta.



Fonte: O Autor.

Inicialmente a interface web envia as informações inseridas pelo usuário para a API, que trata estas informações e as transcreve para uma inserção no banco. Neste momento, como retorno para a API, tem-se o identificador “ID” desta inserção, que em seguida é enviado para a interface como resposta da requisição. Feito isso, o algoritmo de cálculo e ETL é requerido, passando como parâmetro para sua execução o identificador da distribuição a ser calculada. Ao fim, o algoritmo insere a tabela resultante no banco de dados, o qual é lido rotineiramente pela interface web.

3.1 Requisitos Funcionais

Conforme apresentado anteriormente, o projeto deve atender certas funcionalidades, a partir das quais compõem-se requisitos funcionais a serem atingidos. São estes:

- Cálculo de distribuição de peças;

Espera-se que a ferramenta calcule a distribuição de maneira ágil e com resultados precisos.

- Informação do resultado e apresentação de histórico;

Deseja-se que o sistema informe o resultado dos cálculos e apresente os registros das distribuições anteriormente realizadas.

- Possibilidade de edição do resultado final da distribuição;

Como o usuário final trata-se de um profissional com conhecimento da área, certos fatores que não entram no cálculo do algoritmo podem interferir no resultado final da distribuição. Desse modo, levando em consideração estes fatores, o usuário pode realizar um ajuste fino no resultado final da distribuição.

- Geração e exportação das distribuições;

Espera-se que o sistema possibilite a exportação dos resultados das distribuições, através da geração de um arquivo contendo as informações da tabela resultante contendo as quantidades de peças destinadas a cada filial.

- Precisão e robustez no cálculo da distribuição desejada;

Deseja-se que a ferramenta execute o cálculo seguindo a lógica programada, sem perturbações e perdas no fluxo dos dados, para consequentemente apresentar resultados fidedignos com informações coerentes.

3.2 Requisitos Não-Funcionais

Diante das necessidades expostas, os itens a seguir abrangem as informações necessárias para guiar o desenvolvimento do projeto.

- Comunicação com interface web

Para dar início ao cálculo, é necessária a aquisição dos dados informados pelo usuário, assim, é crucial a capacidade de interlocução entre a interface web e o algoritmo.

- Conexão com banco de dados

O sistema deve ter aptidão de conectar-se com o banco de dados para extrair e inserir os dados necessários.

- Aquisição de dados de diferentes fontes

Conforme citado, serão necessários dados de diferentes fontes para os cálculos, desse modo, o sistema deverá apresentar a possibilidade de diferentes conexões para estas aquisições.

- Registro do resultado final

O sistema deverá registrar os resultados encontrados relacionando a correta distribuição demandada.

- Agilidade no cálculo de distribuição;

Espera-se que a ferramenta calcule a distribuição de maneira ágil, visto que um dos maiores problemas levantados pelo cliente sobre o processo de distribuição utilizado fora a demora no processo.

- Usabilidade do sistema;

Para facilitar a utilização do software pelo seu usuário, busca-se automatizar a maior parte do processo antes executado manualmente. Dado que o tempo de trabalho deste funcionário é custoso para empresa, busca-se diminuir o tempo despendido no processo, que antes consumia em torno de horas e exigia várias etapas, como download de arquivos de um repositório central e a espera pelo cálculo ocasionado pela morosidade do software utilizado.

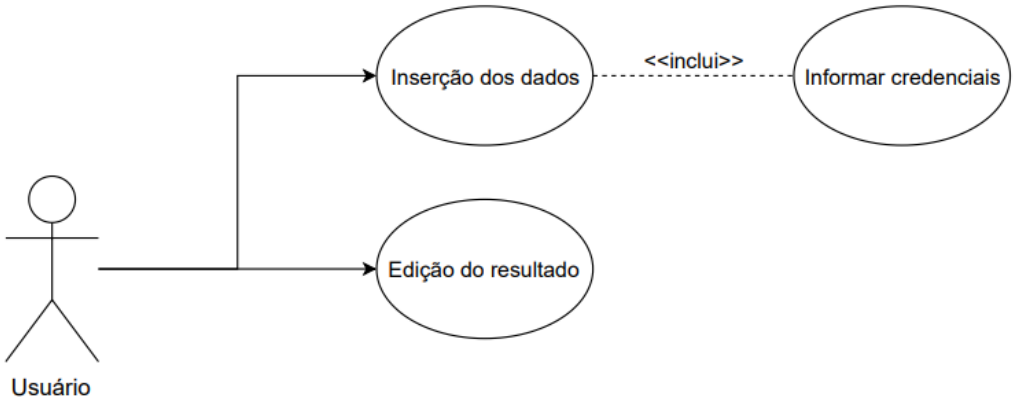
- Manutenibilidade;

Para possibilitar futuras manutenções e atualizações no sistema, é preciso que este apresente características que facilite edições posteriores a entrega, como modularidade e presença de comentários nos algoritmos.

3.3 Modelagem UML

O Diagrama de Casos de Uso representa as funcionalidades do sistema com o meio externo. Para a aplicação desenvolvida, modelou-se o diagrama representado na Figura 6. Neste, tem-se a representação do usuário com os relacionamentos de comunicação com a ferramenta, no caso a inserção dos dados relativos a distribuição de estoque, necessitando de suas credenciais (relacionamento de inclusão) e da edição do resultado disponibilizado.

Figura 6 - Diagrama de Casos de Uso.

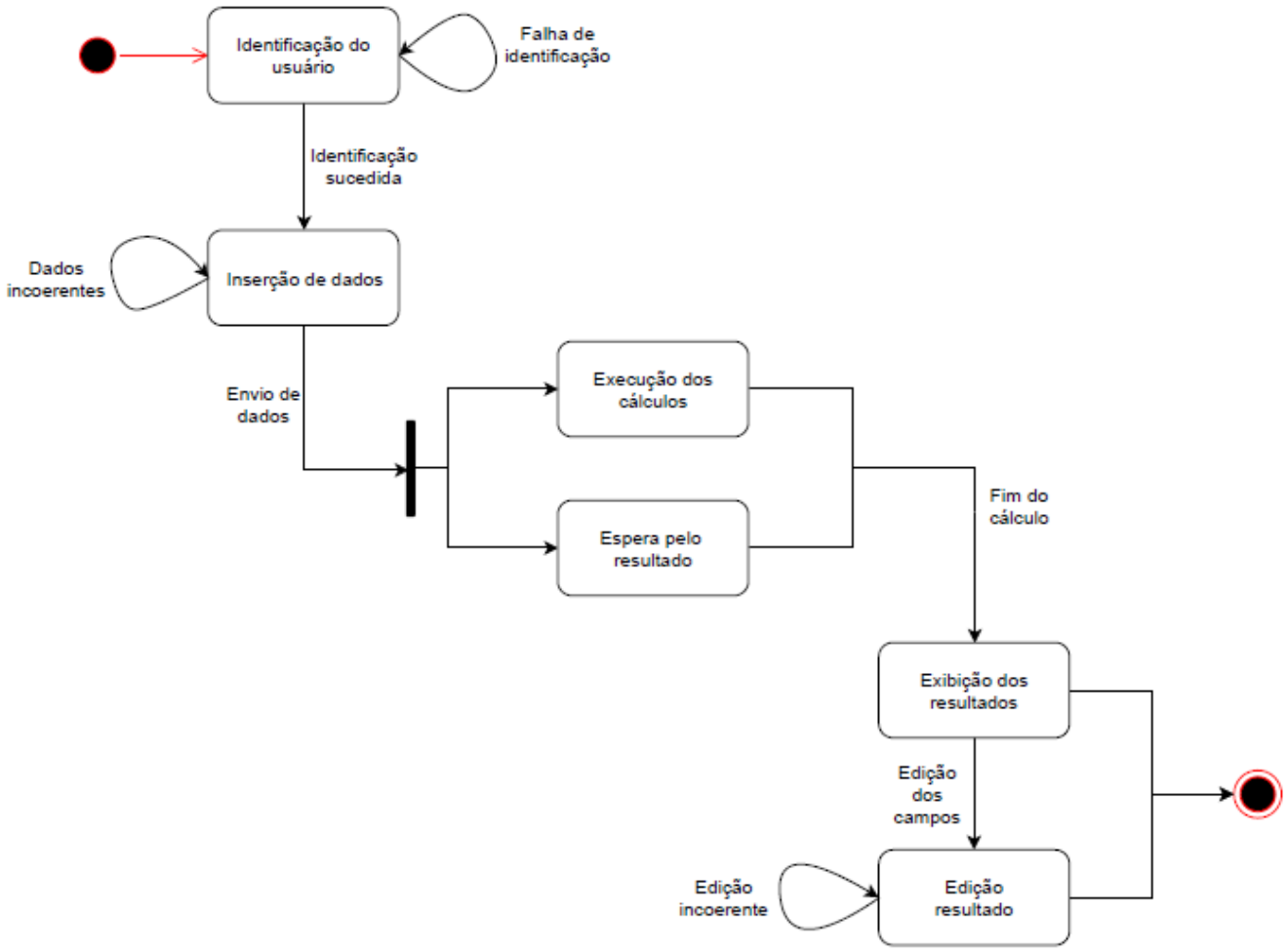


Fonte: O Autor.

Além do citado, optou-se também pela representação do sistema através do Diagrama de Estados. Este, visa representar os estados do sistema e suas transições, a fim de descrever o comportamento dinâmico durante a execução do sistema.

O diagrama é composto principalmente de retângulos, que representam estados, e setas, que descrevem as transições de estado. O evento que aciona cada transição é descrito sobre a seta correspondente. Além disso, esferas marcam o início e o fim do diagrama, sendo a inicial inteiramente preenchida. A Figura 7 apresenta o modelo citado.

Figura 7 - Diagrama de Estados.



Fonte: O Autor.

O estado inicial do sistema se dá com a identificação de seu usuário na interface. Neste momento duas são as possibilidades, as credencias inseridas estão incorretas, o que impossibilita o usuário de seguir, ou estão corretas, permitindo o usuário prosseguir com o uso do sistema provocando a transição para o próximo estado possível. Este, chamado inserção de dados, descreve em seu nome a atividade executada. O sistema permanece no mesmo estado até que os dados relativos à distribuição são preenchidos e confirmados. Assim, o sistema segue para dois estados que ocorrem simultaneamente, um para a interface e outro ocorrendo fora do panorama do usuário. No primeiro tem-se a exibição na interface web sobre o status da execução dos cálculos. O segundo, é a própria rotina de algoritmos do sistema, onde comunicações e cálculos estão sendo executados. Com o fim do cálculo, o estado seguinte passa a ser a

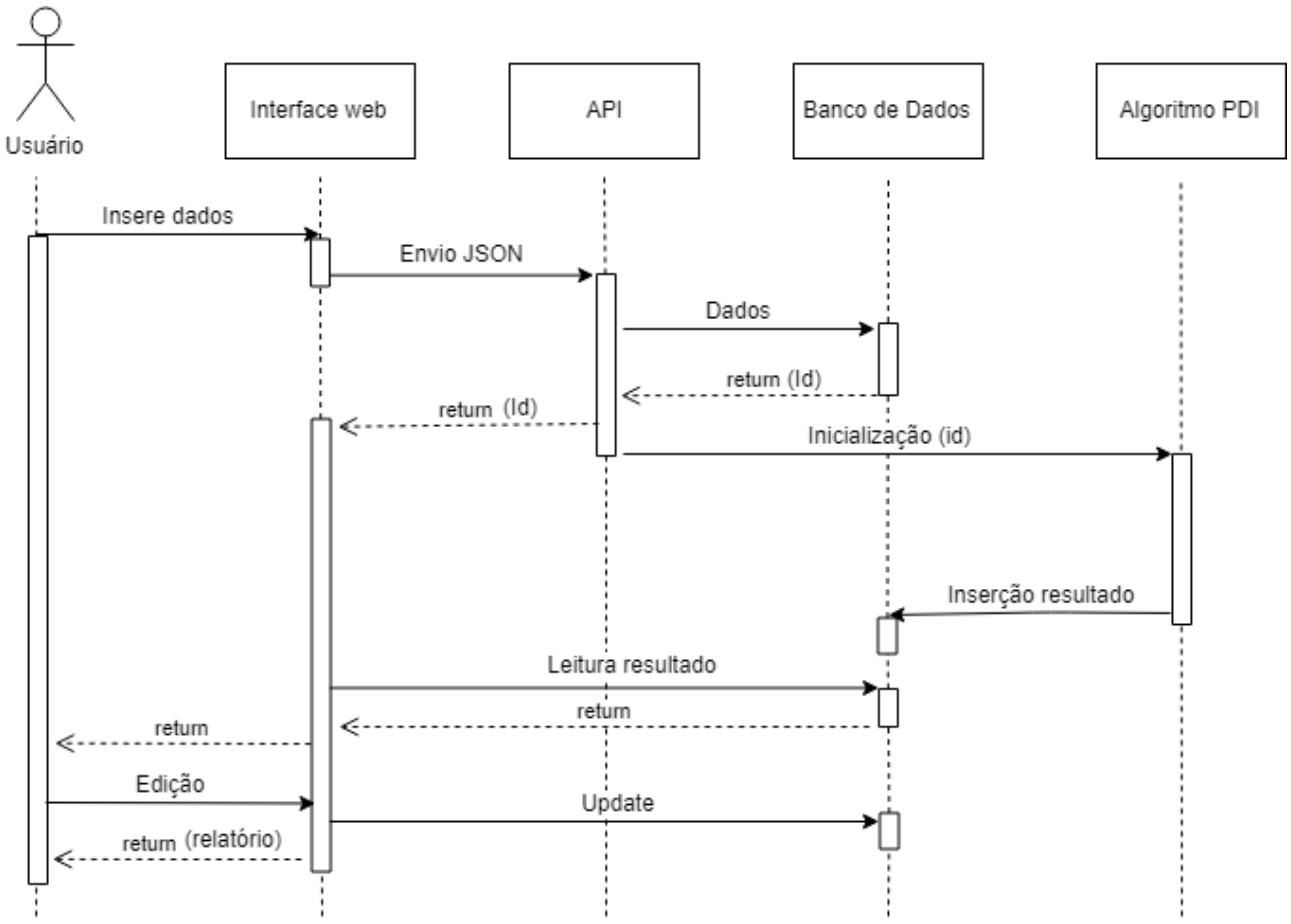
exibição dos resultados, a qual o usuário pode optar por alterar os valores resultantes ou submeter a distribuição com os valores retornados pelo algoritmo.

A comunicação interna deste sistema é aqui detalhada com a representação através de um terceiro diagrama, o Diagrama de Sequência. Neste as interações entre as aplicações que constituem a ferramenta ficam evidenciadas, melhorando a compreensão do funcionamento da aplicação.

Diagramas de sequência possuem dois eixos, o horizontal contendo os objetos envolvidos no fluxo de informação, e o vertical, com a grandeza temporal. Os principais elementos presentes no primeiro eixo, são chamados de objetos, e são representados por retângulos. Neste mesmo eixo pode haver a representação do usuário que inicia a interação, denominado Ator. Cada objeto deste eixo contém uma linha vertical tracejada indicando a execução do objeto. As mensagens trafegadas no fluxo são apresentadas por setas horizontais, incluindo uma descrição de seu conteúdo.

No diagrama de sequência do sistema desenvolvido (Figura 8), a interação é iniciada pelo usuário que, conforme indica a linha continua no seu eixo vertical, acompanha toda a execução sistema. Com a inserção dos dados da distribuição na interface, segundo objeto no fluxo, é iniciado a troca de mensagens do sistema. A interface transforma os dados num arquivo no formato JSON que é enviado para a API pelo método POST do protocolo HTTP. Esta trata os dados recebidos e os insere no banco de dados, o qual retorna esta operação com o uma mensagem contendo o identificador (ID) da tupla adicionada, que pode ser visto no diagrama com a seta tracejada e a descrição “return (ID)”. A API retorna à interface uma mensagem de notificação que o cálculo está sendo processado junto do identificador da distribuição. A inicialização do algoritmo também é ordenada pela API, que passa com o comando o identificador desta distribuição. As delegações realizadas pela API ocorrem não necessariamente nesta sequência, visto que esta opera de maneira assíncrona.

Figura 8 - Diagrama de Sequência.



Fonte: O Autor.

Com a realização dos cálculos e todas as inúmeras lógicas executadas neste processo, o algoritmo PDI insere a tabela resultante no banco através de comando SQL, identificada com o ID presenta nas mensagens trocadas no processo. A interface web, por sua vez, executa a busca a cada período do resultado da distribuição na base de dados através do identificador anteriormente recebido. Quando encontrado retorna à interface estes dados e, caso desejado, o usuário pode altera-los, atualizando a distribuição realizada no banco de dados. Por fim, o ator pode exportar um relatório sobre a distribuição contendo as informações apresentadas na interface.

Os diagramas UML, incluindo os aqui apresentados, demonstram ser ótimas ferramentas de apresentação e elucidação de escopo de projetos. Com a apresentação de funções e interações entre os subprocessos do sistema, tornam compreensíveis aos profissionais envolvidos, analistas ou

desenvolvedores, os detalhes do projeto, minimizando as chances de equívocos e consequentemente retrabalho na elaboração destes sistemas.

4 IMPLEMENTAÇÃO

Nesta seção são apresentadas as ferramentas utilizadas no projeto, seguido de cada parte do processo de concepção do algoritmo e da API desenvolvidos para a ferramenta de distribuição de estoque.

O desenvolvimento partiu da escolha apropriada das ferramentas, seguido da interpretação da lógica contida no antigo sistema utilizado para distribuição do estoque. Com estas informações, pode-se iniciar o desenvolvimento do algoritmo de ETL e *data integration* para os cálculos, seguido do desenvolvimento da WebAPI para troca de mensagens e delegação de rotinas.

4.1 Ferramentas Utilizadas

A adequada escolha da ferramenta de implementação em um projeto é fundamental para atingir bons resultados. Desse modo, nos subcapítulos abaixo serão apresentados maiores detalhes de cada ferramenta utilizada. Vale lembrar que todos os softwares utilizados no desenvolvimento do projeto são gratuitos.

4.1.1 Pentaho Data Integration

Pentaho Data Integration, ou PDI, é um *case open source* da suíte Pentaho, desenvolvida desde 2004 pela Pentaho Corporation e atualmente pertencente a empresa Hitachi Vantara. O software, agora em sua versão 8.2, tem foco no processo de *Business Intelligence,* cobrindo áreas de ETL, reporting, OLAP (Online Analytical Processing), e *data-mining*, além de integração com *Big Data* (HITACHI VANTARA CORPORATION, 2018).

O PDI é o componente mais popular de toda a suíte e isto se deve por se tratar de uma ótima solução para o processo que traz em seu nome: o Data Integration. *Data Integration* é o processo de combinar dados de diferentes fontes, resultando numa vista única destes dados (LENZERINI, 2002). Este é um problema recorrente nas organizações atuais, uma vez que o armazenamento de dados oriundos de diferentes fontes, setores e aplicações é constante.

As fontes de dados em alguns casos podem ser independentes, o que na prática resulta em problemas devido estas serem mutuamente inconsistentes. Apesar disso, procura-se atributos os quais integram estes dados, de modo que estabeleça conformidade entre diferentes fontes.

Neste contexto, o PDI possibilita executar inúmeras operações de integração de dados através de suas diferentes aplicações, que serão apresentadas na sequencia deste documento.

4.1.1.1 Spoon

Spoon é a interface gráfica de edição do PDI. Nela são disponibilizadas funcionalidades avançadas combinadas à fácil programação, permitindo ao usuário implementação e obtenção de resultados satisfatórios sem a necessidade da escrita de códigos. Isso se deve ao modelo ser gráfico visual e à escrita de código ser somente uma alternativa quando se deseja funcionalidades em particular, como no caso de alguns *steps* que apresentam campos para edição de algoritmos em Java Script e SQL.

Na ferramenta é possível criar *Transformations* e *Jobs*, diferentes arquivos que se complementam numa rotina de ETL. *Transformations* são usadas para descrever o fluxo dos dados no processo de ETL. *Jobs* tem como função coordenar sequencias de diferentes *transformations,* executando uma a uma por completo, além de checar condições para execução destas, permitindo a automação de rotinas. A Tabela 2 abaixo representa um comparativo entre as principais funcionalidades destes dois tipos de arquivos.

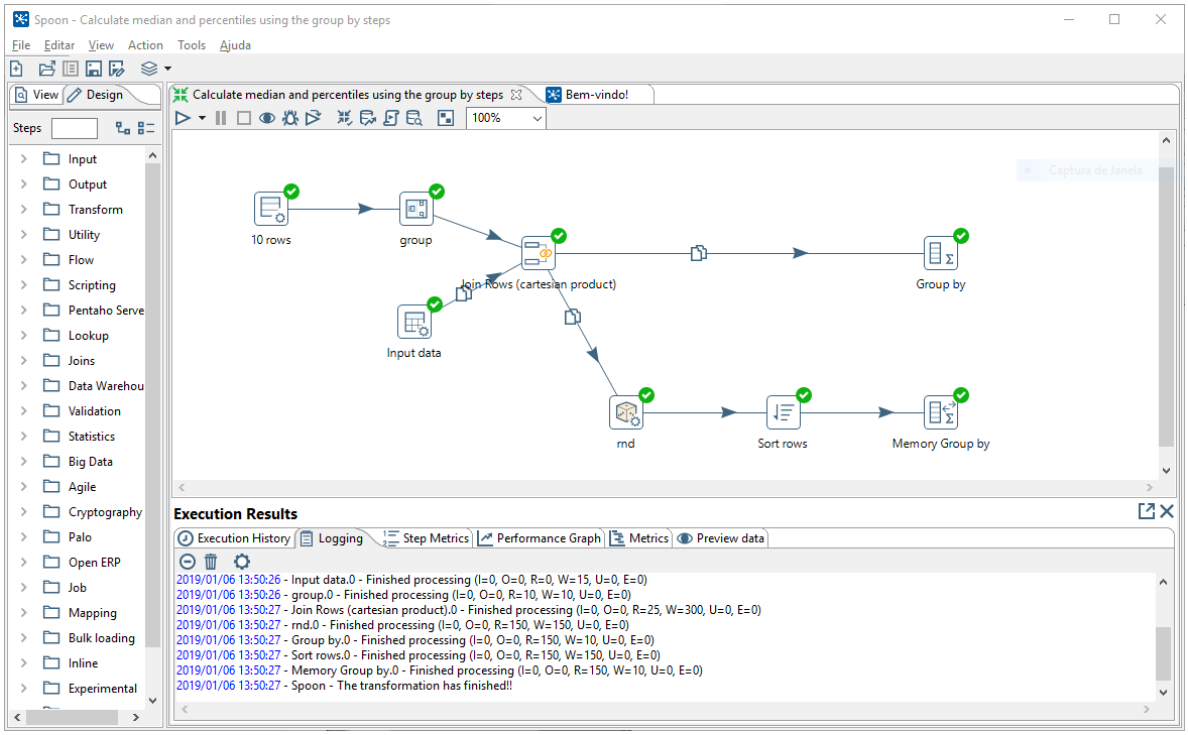
Tabela 2 - Funcionalidade *Job* x *Transformation*.

Job	**Transformation**
Executado sequencialmente	Executado simultaneamente
Opera sobre o fluxo de ações	Opera sobre linha de dados
Organização	Transformação
Testar condições	Aplicar regras e lógicas

Fonte: Adaptado de CESARIO, COSTA e DE SALLES (2017)

Na Figura 9 tem-se uma representação da interface de edição. Na aba esquerda pode-se ver os *steps* disponíveis para edição das rotinas, separados por grupos com diferentes funcionalidades. No centro, tem-se o painel de edição, denominado Canvas, e, abaixo, o Painel de execução apresenta todas as informações necessárias sobre a tarefa executada, incluindo métricas de cada *step*, performance e visualização prévia dos dados.

Figura 9 - Interface Spoon.



Fonte: O Autor.

4.1.1.2 Pan e Kitchen

Enquanto Spoon apresenta a interface para edição dos algoritmos de ETL, o Pan e Kitchen são as aplicações que executam as operações via linha de comando, ou seja, nos ambientes de produção, sem supervisão humana (BACKER, 2015).

Enquanto o Pan executa os arquivos de *transformations* caracterizados pelo término em “.ktr”, o Kitchen executa as *jobs*, terminados em “.kjb”. Ambos podem executar aplicações a partir de um sistema de arquivos ou de repositório de banco de dados.

No geral, quando em produção, suas execuções geralmente são agendadas através de gerenciadores de tarefas dos sistemas operacionais, por exemplo, através da execução de arquivos em lotes, como “.bat”.

No caso deste projeto, utiliza-se da aplicação Kitchen para a execução do algoritmo no *background* do servidor. Apesar de ser composto de diferentes *transformations,* estas fazem parte de uma *main job* que sincroniza suas execuções. O comando para o início da execução do algoritmo pelo Kitchen fica sobre o cargo da WebAPI.

4.1.2 MySQL

MySQL é um servidor e gerenciador de banco de dados relacionais, ou RDBMS (*Relational Database Management Systems*) *open source,* com base em SQL. Projetado inicialmente para trabalhar com aplicações de pequeno e médio porte, passado por inúmeros avanços e desenvolvimento de novas versões, atualmente atende também aplicações de grande porte, tendo o rápido acesso como seu diferencial (MILANI, 2007).

Por possuir a confiabilidade como uma de suas características, o MySQL é utilizado em aplicações de todas as áreas de negócios, além de aplicações intensas para a internet, com acessos e requisições a todo o tempo, como em lojas virtuais e soluções Web (MILANI, 2007).

Os dados trabalhados neste projeto, são hospedados em bancos de dados gerenciados por este sistema. Constantemente submetido a requisições, tem suma importância para a realização de testes e consultas sobre os dados utilizados na concepção deste projeto, além da verificação de seus resultados, visto que é integrado de uma interface para a visualização das tabelas.

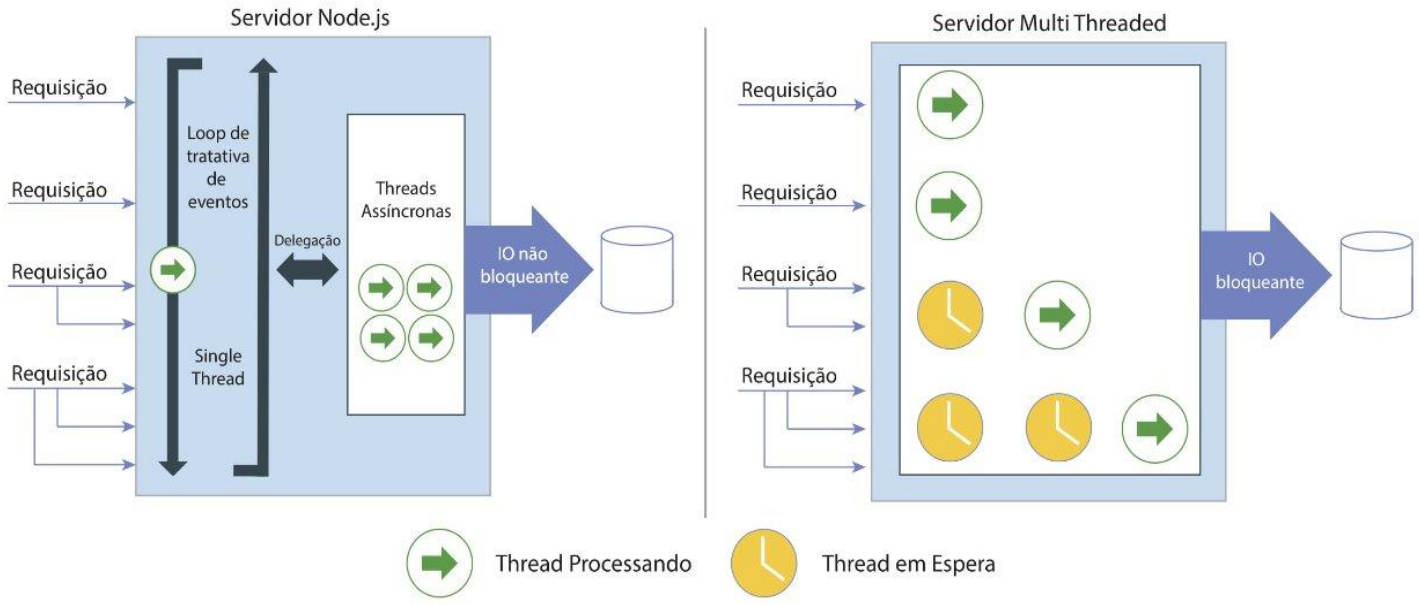
Outra importante funcionalidade deste SGDB é o auto incremento de valores em uma coluna. Também conhecido pelo seu comando “AUTO_INCREMENT”, a funcionalidade nativa do MySQL gera um novo valor para cada nova tupla adicionada na tabela, neste caso, identificando e garantindo a unicidade de cada nova distribuição realizada.

4.1.3 Node.js

Node.js é um interpretador *open source* de JavaScript projetado para criar aplicativos de rede escalonáveis (NODE.JS FOUNDATION). Focado em aplicações *server-side,* esta ferramenta opera em modo assíncrono e orientado a eventos, o tornando uma solução leve e ideal para aplicações em tempo real com troca de dados.

Apresentado em 2009, pelo seu criador Ryan Dahl, esta plataforma tornou-se rapidamente popular ao solucionar o problema da manipulação dos servidores com o grande número de conexões concorrentes em seus programas. Enquanto que tradicionalmente cada conexão com o servidor cria uma nova *thread* e aloca uma certa quantidade de memória a esta, fazendo com que a quantidade de conexões se limite a memória RAM existente na máquina, o Node.js usa o modelo de I/O direcionada a evento não bloqueante, fazendo com que cada nova conexão dispare um evento executado dentro da *engine* de processos da plataforma (NODEBR COMMUNITY, 2016). Esta comparação é ilustrada abaixo, na Figura 10.

Figura 10 - Comparação entre os servidores.



Fonte: Adaptado de ROTH (2014).

Outros fatores positivos do Node.js são sua leveza, diminuindo os custos de hardware do servidor hospedeiro, ser multiplataforma e de usar a linguagem JavaScript, unificando a linguagem de aplicações web sendo usada tanto no *front-end* como no *back-end* e facilitando a comunicação entre estes dois lados. Entre as principais aplicações desta plataforma destaca-se a criação de APIs, aplicações em tempo real, *back-end* de jogos e aplicativos de mensagens (CHANDRAYAN, 2017).

Neste projeto, este interpretador dá o suporte necessário para o funcionamento da web API hospedada no servidor da empresa cliente. Considerando este um hardware de extrema relevância para todos os sistemas virtuais em atividade, é relevante minimizar sua alocação de memória. O que confere com as características do Node.js.

4.1.3.1 Express

O uso de frameworks para o desenvolvimento de aplicações fomenta o desenvolvedor com funcionalidades que vão além da linguagem nativa. Neste contexto, o Express foi lançado para dar suporte para o Node.js na criação de aplicações web e APIs (NODE.JS FOUNDATION, 2017).

Muitas vezes, os desenvolvedores de aplicações tendem a lidar com rotinas de algoritmos repetitivas, detalhistas e limitadas de recursos. Desse modo, frameworks como Express, tornam este processo muito menos custoso, agregando funcionalidades e poupando a escrita de códigos repetidos, resultando num desenvolvimento muito mais descomplicado.

Segundo HAHN (2016), o Express traz dois grandes recursos ao servidor Node.js: adiciona conveniencias uteis ao servidor HTTP do Node.js, abstraindo a complexidade do código (ex.: enviar uma imagem JPEG com Node.js bruto é complexo, com Express basta uma linha); e permite fatorar uma função de manipulador de solicitação monolitica em vários manipuladores de solicitações menores, operando na casa de bits. Com isso o processo se torna mais sustentável e modular.

Desse modo, o Express auxilia o desenvolvedor a ter seu trabalho focado na implementação das tarefas de sua aplicação e menos em detalhes técnicos da linguagem usada.

4.2 Interpretação dos cálculos

A empresa cliente, quando apresentou a necessidade da automação do cálculo da distribuição de seu produto para as filiais, nos forneceu a ferramenta até então utilizada. Esta era tida como uma “caixa preta” pelos seus usuários, levantando a necessidade de investiga-la e, em seguida, documentar as informações nela contida para os cálculos das distribuições.

No processo original era necessário inicialmente o download de uma planilha contendo o estoque de produtos apresentado nas duas centrais de distribuição na data em questão para, então, seus dados serem extraídos pela ferramenta da máquina do usuário. A partir das informações descritas por este nos campos disponíveis, o cálculo iniciava.

O resultado final apresentado pela ferramenta era obtido de uma escolha entre dois cálculos principais: por histórico de venda ou por velocidade de venda. Além disso, visando a implementação futura, levantou-se o mapeamento das fontes dos dados necessários, incluindo nomes de tabelas e campos utilizados.

Estes serão brevemente descritos abaixo, com abstrações de maiores detalhes devido as fórmulas apresentarem informações com certo sigilo da empresa cliente.

4.2.1 Cálculo por histórico de venda

Um dos importantes dados contidos no banco da empresa é relacionado ao histórico de vendas de cada linha de produto por cada filial. Esta informação é proveniente do banco de dados central do software de ERP da empresa e é atualizada periodicamente.

Uma das informações iniciais descritas pelo usuário é o percentual de estoque disponível de cada centro de distribuição que se deseja emitir. O produto

destas informações resultada na quantidade a ser distribuída considerando o histórico de vendas, como segue:

$$Valor\ por\ histórico = estoque\ disponível \cdot \%\ de\ venda\ da\ filial \cdot \%\ estoque\ a\ distribuir$$

O valor resultante deste cálculo, para cada filial, nem sempre é um valor exato e, sendo o produto da empresa peças de alta exclusividade e pouquíssimos exemplares iguais, o destino de cada unidade é de grande importância. Com isso, os fracionários de cada resultado são distribuídos num cálculo iterativo entre todas as filiais para, ao fim, ter-se valores inteiros para cada destino. Maiores detalhes deste cálculo serão descritos nos capítulos seguintes.

4.2.2 Cálculo por velocidade de venda

Assim como para o cálculo anterior, são utilizadas informações inseridas pelo usuário, bem como já armazenadas no banco da empresa. Neste caso, o cálculo é produto da velocidade de vendas de cada produto por filial e do lead time de entrega do centro de estoque para esta.

No caso do lead time, é considerada a relação de cada filial com o centro de distribuição geograficamente mais acessível para o recebimento dos produtos. Já a velocidade de venda é extraída também do sistema de gestão da empresa, o qual contabiliza o tempo que cada linha de peça permanece na filial até ser vendida. Ambos os dados estão em função de dias.

$$Valor\ pro\ velocidade = lead\ time \cdot velocidade\ de\ venda$$

Pode-se interpretar este resultado como a quantidade de peças que se necessita repor na loja para o intervalo de tempo da entrega dos produtos até a mesma.

4.3 Desenvolvimento do algoritmo

Nos subcapítulos seguintes serão apresentadas as etapas do desenvolvimento do algoritmo de ETL.

4.3.1 Escolha da Ferramenta de ETL

Feita a definição dos requisitos de projeto, buscou-se atender as necessidades do usuário com a melhor opção de ferramenta disponível. A empresa Bix Tecnologia é especialista em sistema de *business intelligence,* tendo grande domínio do uso das ferramentas Qlik View e Qlik Sense, as quais, juntamente com Pentaho, ocupam grande parte do mercado do ramo.

Com isso, os projetos preferencialmente são desenvolvidos com o uso das ferramentas Qlik, pois além do grande conhecimento dos funcionários da empresa, a ferramenta apresenta maiores facilidades de uso e implementação quando comparada aos concorrentes, tornando o processo de desenvolvimento mais rápido e menos dispendioso.

Apesar disso, a licença para seu uso é cobrada por cada usuário da ferramenta e, por ser voltada inteiramente ao *business intelligence*, sua interface não permite edições das informações em seus campos, como edição dos valores. Isto contradiz um dos importantes requisitos solicitados pelo cliente, que é o ajuste fino dos resultados pelo usuário da ferramenta. Com isso, outra opção que também se mostrou atrativa é a do software Pentaho em conjunto com o uso de uma interface externa, neste caso, uma página web já existente. Além de ser uma ferramenta de uso abrangente, sua licença é gratuita, sem custos para seu usuário.

Sua programação se dá através de blocos interligados, similar a um fluxograma. Cada um destes blocos, denominados *steps*, executam funções especificas sobre os dados, permitindo inúmeras tratativas sobre estes. A programação de cada etapa do algoritmo será descrita nos capítulos seguintes.

4.3.2 Extração dos dados

Conforme citado no subcapitulo 2.2, a primeira etapa em uma ETL é a extração. Nela os dados são coletados de suas diferentes fontes para posteriormente serem transformados e carregados.

No desenvolvimento do projeto, trabalhou-se inicialmente com extração proveniente de arquivos de planilhas estáticas, protegendo os bancos dos testes do algoritmo. Neste momento, as planilhas eram fornecidas pelo cliente e apresentavam os mesmos campos e tipos de dados de uma tabela do banco de produção.

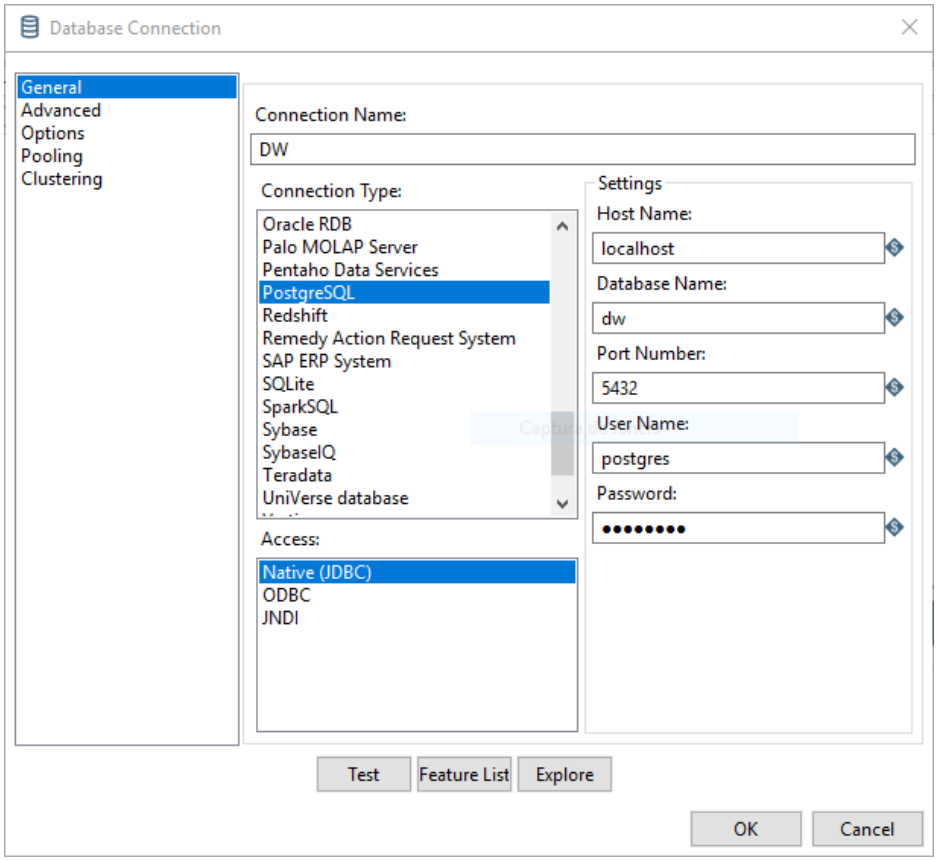
O processo em Pentaho para a extração de dados de planilhas é exibido na Figura 11 abaixo. Inicialmente, o bloco denominado "*Table Input"* extrai os dados da planilha e o seguinte atualiza a tabela no banco de dados com os novos dados. O processo de armazenar numa tabela intermediária, para ser utilizada na etapa seguinte de transformação, facilita a reutilização dos mesmos dados em diferentes partes do algoritmo, otimizando o processo

Figura 11 - Extração.

Já para extração de tabelas do banco de dados a sequência lógica é a mesma, porém uma conexão com o banco em questão deve ser anteriormente definida. O Pentaho permite inúmeras conexões de diferentes fontes, como forte característica do *Data Integration,* conforme descrito em 4.1.1. As possibilidades incluem bancos comercialmente difundidos, como Oracle, bancos de código aberto, como PostgreSQL, e modernos bancos de dados, além de permitir, ainda, adicionar conexões com bancos que não estão presentes na lista. Para isso, basta adicionar o driver conector JDBC (*Java Database Conectivity* – aplicação Java de conexão com banco de dados) para o banco em questão e adiciona-lo ao diretório de instalação do PDI.

A janela de configuração da conexão pode ser vista na Figura 12. Para proteger dados sigilosos do cliente, utilizou-se no exemplo dados fictícios. Nesta, é necessário adicionar um nome à conexão, seguido da escolha do tipo de conexão (janela *Connection Type*). Feito isso, no painel abaixo (*Access*) é selecionado o método de acesso ao banco e, então, as propriedades da conexão devem ser inseridas nos campos do painel *Settings.* Ao fim, pode-se executar um teste da conexão ao clicar no botão *Test*, apropriado para se evitar impasses com a conexão em etapas seguintes.

Figura 12 - Configuração da conexão com banco.

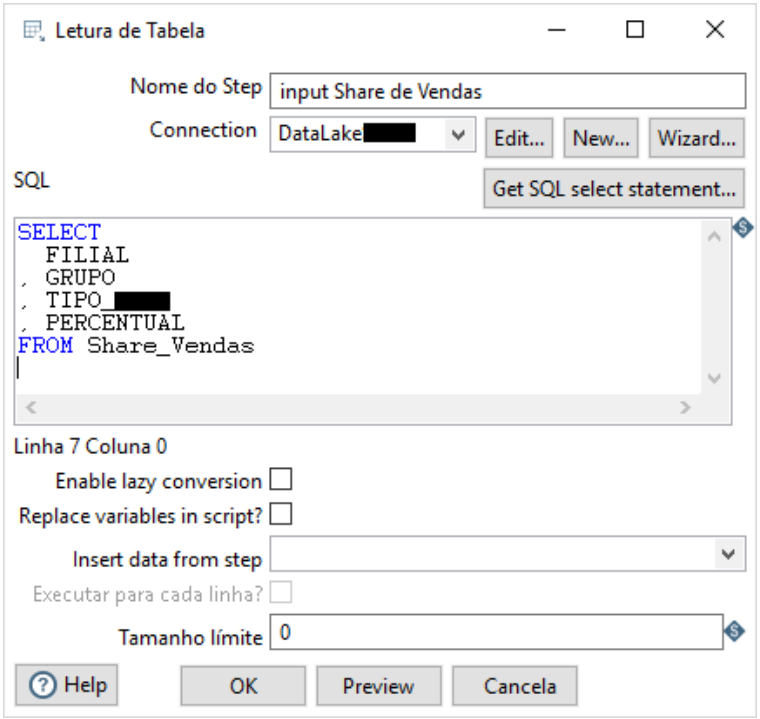


No propósito deste projeto, a implementação da ETL foge do comumente aplicado em sistemas onde este processo é mais usual, como aplicações puramente de *business intelligence*. Neste caso, lógicas para a comunicação entre os processos operarem corretamente exigiram etapas incomuns, porém as quais o PDI sustentou satisfatoriamente com sua infinidade de *steps* voltadas ao *Data Integration*.

A distribuição de estoque entre as filiais da franqueadora se dá partindo das informações lidas no banco de dados, as quais foram informadas pelo usuário na interface web, e então inseridas no banco pela web API (conforme mostra o esquemático da Figura 5). O sistema, ao ser requisitado por esta, recebe em uma variável já declarada o ID da linha da tabela que contém estas informações da distribuição a ser realizada. Com as informações presentes no banco, a extração dos dados pode ser realizada.

Feito isso, a configuração para uma extração de uma tabela de banco pode ser vista na Figura 13. O caso representado na figura, é uma das inúmeras tabelas extraídas do algoritmo, e é a responsável pela leitura dos dados relativos ao histórico de venda das filiais. Para uma extração do banco de dados, comandos SQL são passados na janela do *step,* e serão executados no andamento da aplicação. Deve-se também especificar o banco de origem da tabela, no campo "*Connection*".

Figura 13 - Extração do banco de dados.



Realizada a coleta dos dados, pode-se armazenar o resultante em uma tabela ou planilha especifica e então seguir para etapa de transformação, a qual

se pode aplicar uma infinidade de tratativas aos dados. No caso deste projeto, ambos os casos são presentes.

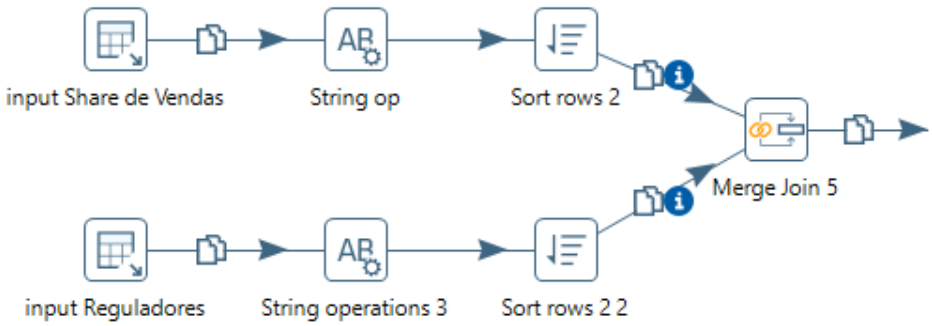
4.3.3 Transformação dos dados

Carregados os dados que serão utilizados no algoritmo, a etapa seguinte é a de transformação destes dados. Neste projeto, esta etapa conta com os diversos cálculos das distribuições, uniões entre tabelas e algoritmos em JavaScript e Python para auxiliar na execução da lógica.

De modo geral, busca-se unir os dados necessários para os cálculos numa tabela central, o que, para diversas arquiteturas, nomeia-se tabela “Fato”. Como apresentado no subcapitulo 4.1.1 Pentaho Data Integration, separa-se as aplicações criadas no Pentaho em *transformations* e *jobs*, neste caso, cada um destes processos de criação da tabela central seguido dos cálculos será realizado em uma *transformation* específica relacionada ao tipo de distribuição em questão.

Para isso, logo na sequência do *step* de *Input,* realiza-se a união entre as tabelas desejadas. Como premissa deste *step*, os atributos chave nas tabelas devem estar ordenados, fazendo necessário o uso do *step “Sort Rows”.* A sequência descrita é exibida na figura abaixo.

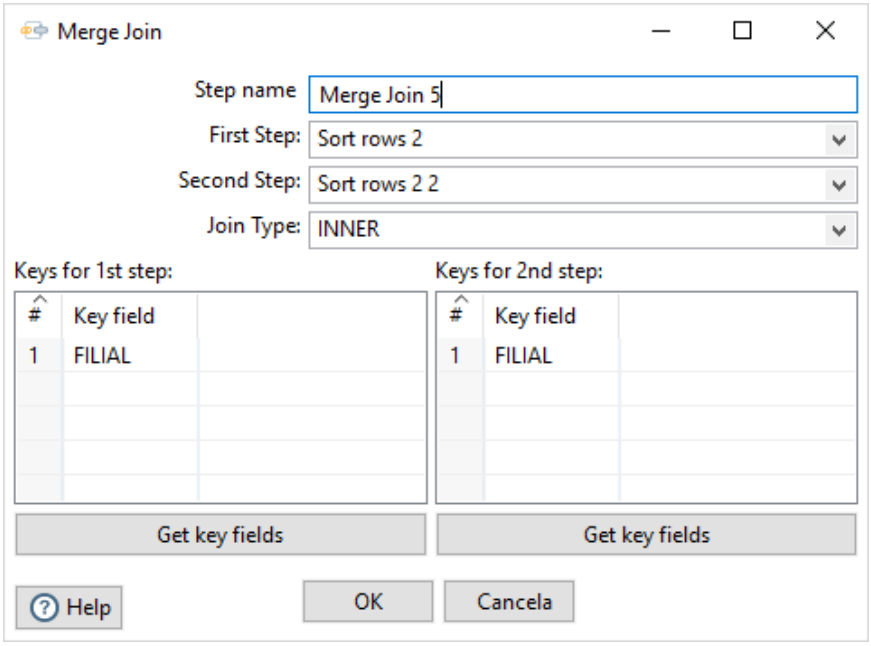
Figura 14 - União entre tabelas.



As configurações do *step* “*Merge Join”* são apresentadas abaixo. Nos campos de edição, deve-se especificar os blocos anteriores a etapa para correlacionar com a descrição das chaves, o que é feito nas janelas paralelas

centrais. Abaixo, na Figura 15, é especificado o tipo de *Join* que se deseja executar, podendo ser INNER, LEFT OUTER, RIGHT OUTER, e FULL OUTER.

Figura 15 - *Merge Join step*.



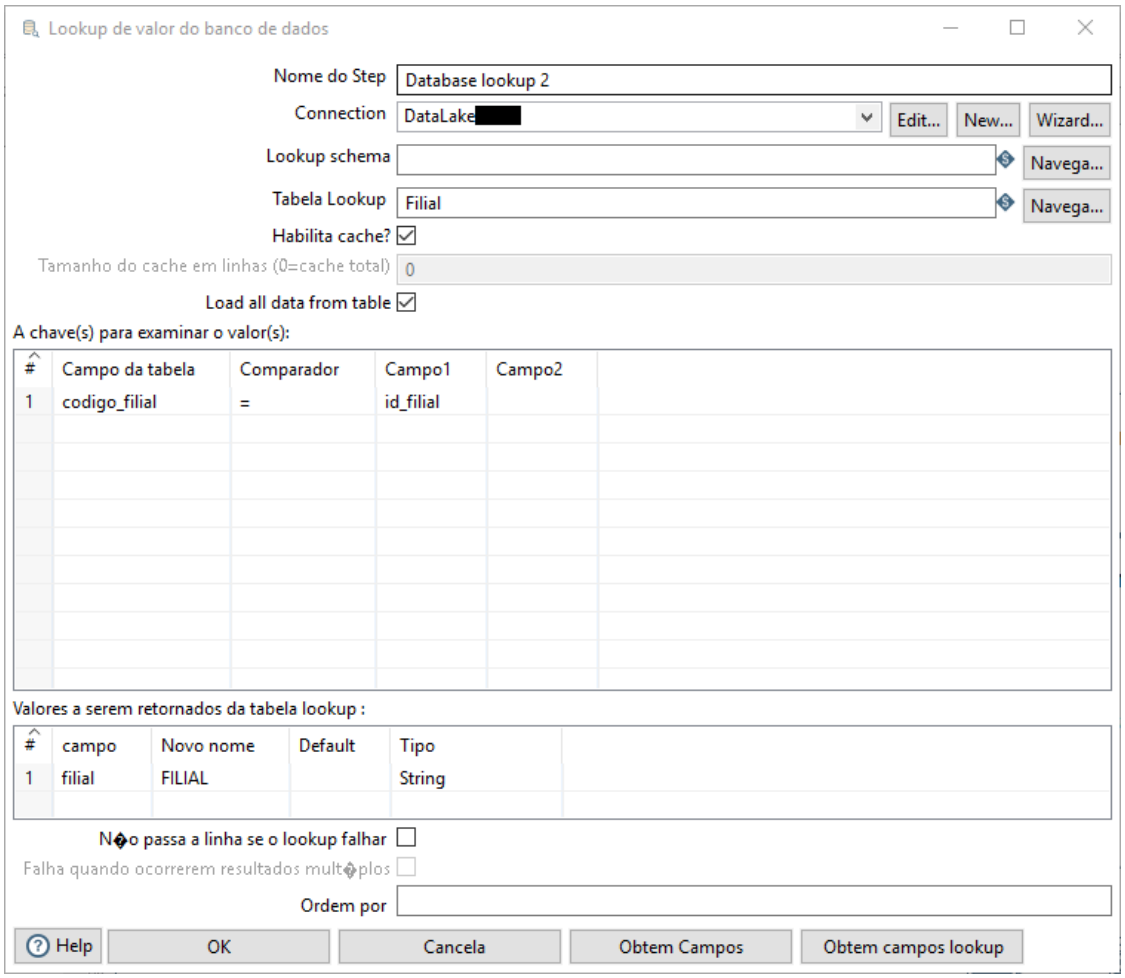
Este processo, muitas vezes acarreta em inúmeros *steps* intermediários, visto que em alguns casos os dados que se pretende unir não tem correlação direta, ou seja, não apresentam um atributo em comum entre as tabelas. Assim, é necessário compor chaves, concatenando diferentes atributos, a fim de se obter um campo único em comum.

Outro *step* muito utilizado na lógica da transformação é o "*Database lookup"*. Este tem como função analisar em uma tabela especificada um certo campo e, a partir de uma comparação lógica, retornar um atributo desta tabela alvo.

A Figura 16 representa a janela de configuração do bloco para o exemplo onde deseja-se buscar o nome das filias na tabela "Filial" a partir do identificador da filial presente na tabela do algoritmo, visto na captura como "id_filial", presente no "Campo1". Para isso, além de especificar a conexão com o banco e a tabela que será analisada, deve-se colocar o nome dos atributos envolvidos na comparação e o atributo que será retornado em caso de a comparação ser satisfeita. No exemplo, caso a comparação entre os campos "id_filial" e o campo

da tabela “codigo_filial” seja verdadeira, o atributo “filial” será adicionado à tabela do algoritmo, conforme especificado na janela inferior da captura.

Figura 16 - *Database lookup*.

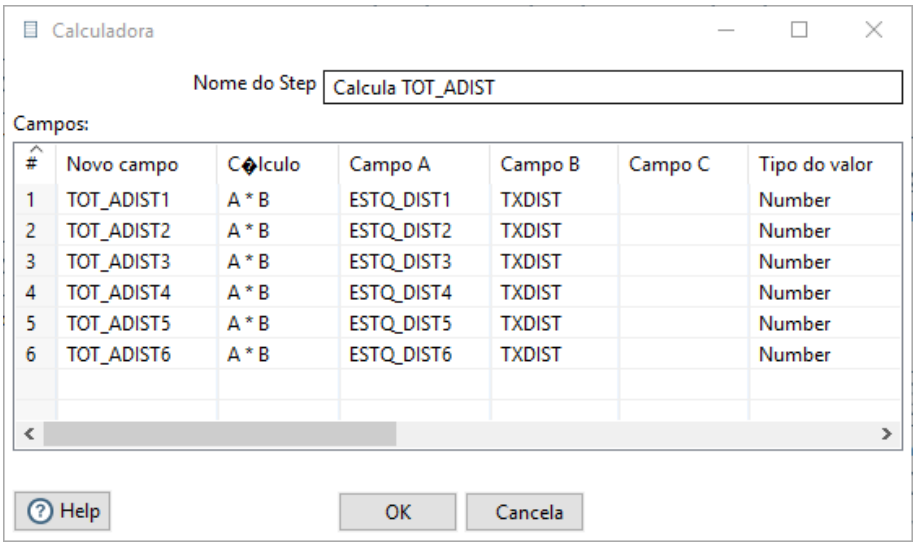


Utilizando das etapas apresentadas até aqui, tem-se centralizado numa tabela os atributos e dados necessários para os cálculos. Em seguida, para a execução destes, são utilizados diferentes *steps*.

Um destes, o *step “Calculator”,* como o nome sugere, executa algumas operações matemáticas pré-estabelecidas entre dois atributos especificados, gerando um terceiro contendo o resultado da operação. A janela de configuração é exibida na Figura 17, na qual se declara os nomes dos campos envolvidos na operação, o nome do atributo que conterá o resultado desta e a operação selecionada. Neste exemplo, parte da distribuição por histórico, executa-se o cálculo do total a distribuir “TOT_ADIST” multiplicando o percentual que se

deseja distribuir relativo ao total disponível em estoque, ou seja, a taxa de distribuição “TXDIST”, pelo estoque disponível pra distribuição “ESTQ_DIST”.

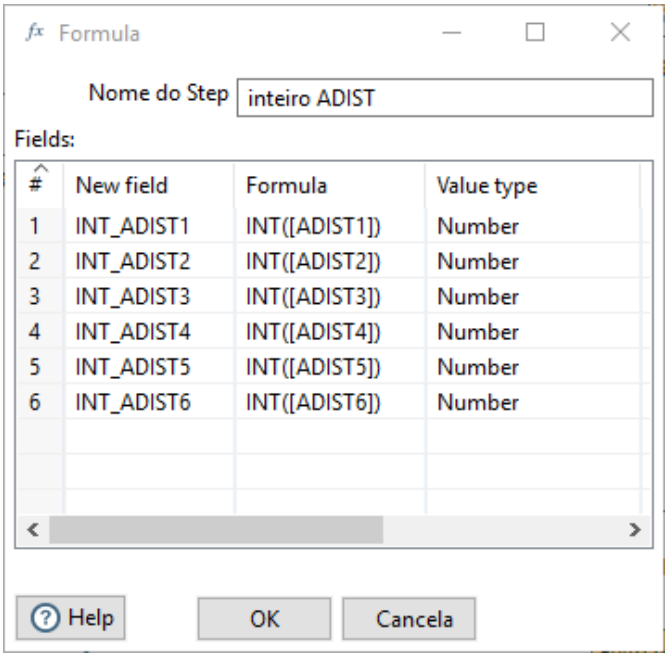
Figura 17 - Configuração do *step Calculator*.



Similar ao bloco anterior, porém permitindo maior liberdade de edição dos cálculos, o *step* “*Formula*”, é capaz de realizar inúmeras operações entre atributos especificados e presentes numa mesma tabela, gerando um novo atributo com o resultado da operação programada. Dentre as fórmulas possíveis, tem-se operações matemáticas básicas como soma, subtração, divisão e produto, e de comparações. Além disso, é possível programar expressões condicionais, com operadores como *if*, *or* e *and*; e de formatação de datas, especificando o tipo de data que se deseja formatar o dado em questão.

No exemplo abaixo, Figura 18, a utilização deste *step* resulta no atributo contendo a parte fracionária do resultado da distribuição do cálculo por histórico de venda. Para isso, utiliza-se a operação matemática “INT”, que seleciona a parte inteira do valor do atributo e atribui ao novo campo que está sendo criado. Esta etapa é importante para o seguinte cálculo da parte fracionaria deste valor, visto que cada unidade das peças que estão sendo calculadas é de suma importância no resultado final, conforme descrito no capítulo anterior.

Figura 18 - Exemplo *step Formula*.

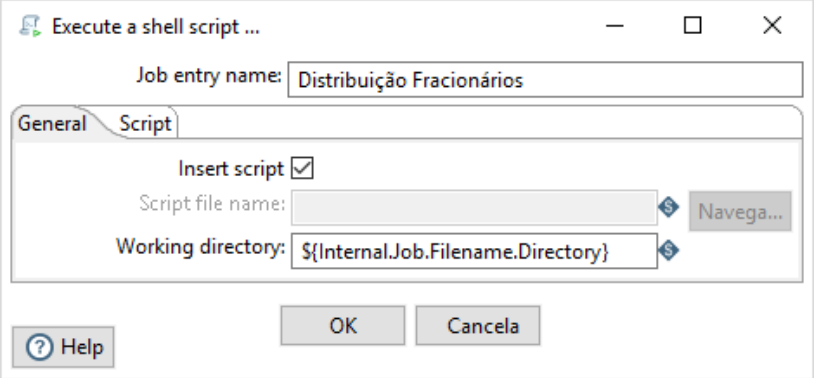


A etapa seguinte à do exemplo anterior, a de distribuição dos fracionários, também faz uso de um *step* proficiente permitido pela ferramenta Pentaho. Nesse, chamado "*Shell*", é possível executar um arquivo que contenha comandos a serem executados pelo prompt de comando da máquina hospedeira da aplicação.

No caso deste projeto, foi implementado um algoritmo em linguagem Python para a execução do algoritmo interativo de distribuição da parte fracionária resultante dos cálculos do algoritmo entre as filiais envolvidas na distribuição. Como resultado, tem-se a exata quantia designada a se distribuir, partilhada em valores inteiros entre as filiais da distribuição. O algoritmo Python pode ser visualizado no Apêndice A. Neste, foi utilizada a biblioteca Pandas, voltada para análise de dados que facilita a manipulação de dados de tabelas. O *output* do algoritmo é uma tabela contendo o resultado final da distribuição por histórico, executado a distribuição dos fracionários.

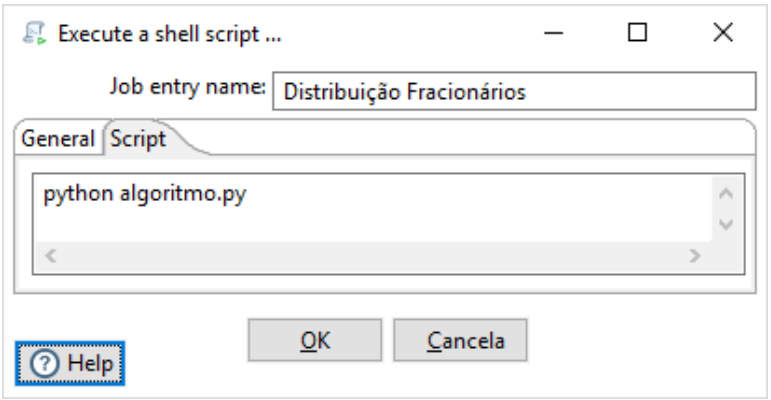
As configurações do *step Shell* podem ser vistas abaixo, na Figura 19. No campo editável, coloca-se o diretório do arquivo contendo os comandos, neste caso, representado pelo conteúdo da variável que contém o caminho do diretório do próprio algoritmo.

Figura 19 - Configuração do *step* shell.



Na aba "*Script*", exibida na Figura 20, tem-se o comando que será executado no prompt, que no caso solicita a execução do arquivo que contém o algoritmo, utilizando do interpretador Python.

Figura 20 - Aba Script *step Shell*.

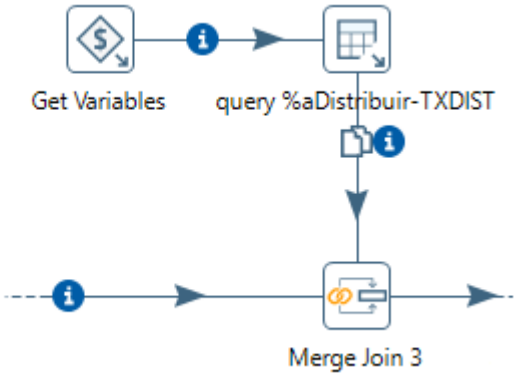


Basicamente, para os dois algoritmos utilizou-se das sequências lógicas já apresentadas para resultar na tabela central com os atributos calculados, diferenciando em poucos *steps*.

Um exemplo deste caso, no cálculo por histórico, parte da lógica é diferente devido ao algoritmo necessitar da taxa de distribuição usada nos cálculos. Para isso, é necessário ler do banco de dados o campo específico proveniente das informações iniciais do usuário na interface web, que a API insere no banco. Visto que, ao iniciar o algoritmo, é passado o identificador da tupla do banco referente à distribuição que está sendo feita, esta informação é armazenada numa variável no ambiente do Pentaho e pode ser requisitada dentro das transformações através do *step "Get Variables"*. Tendo esta, basta

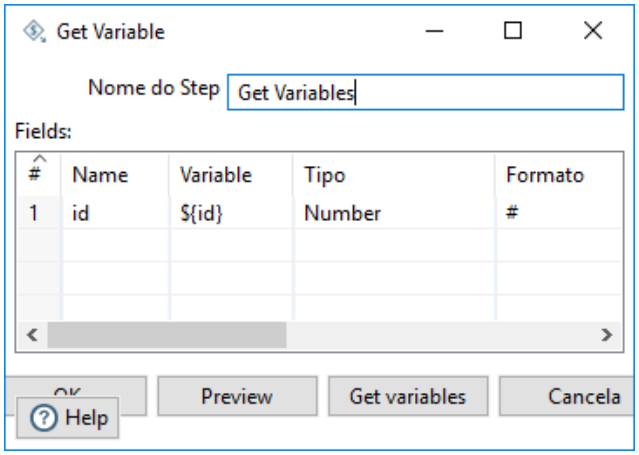
consultar a tabela com o *step* “*Table Input*” especificando o campo de leitura através do identificador. Feito isso, o resultado da consulta pode ser inserido na tabela central de dados, através do bloco “*Join*”, também apresentado anteriormente. Esta sequência é apresentada na Figura 21.

Figura 21 - Sequência para leitura de dado a partir de variável.



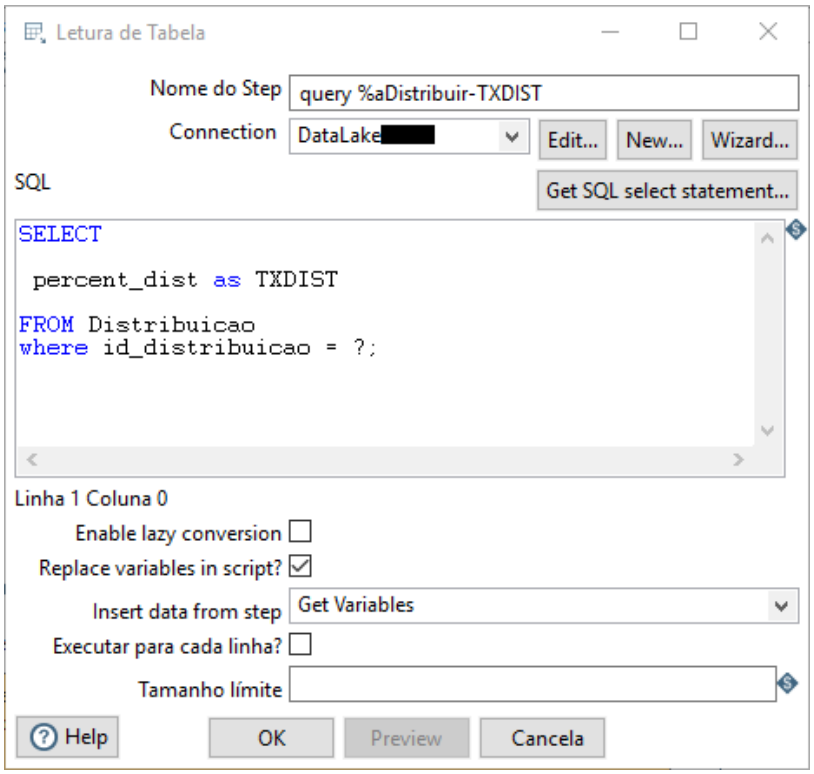
Na Figura 22 - Step Get variable., tem-se a configuração do *step* “*Get variable*”, na qual a variável “id” recebe o identificador presente na variável de mesmo nome declarada no início da ETL.

Figura 22 - *Step Get variable*.



Na sequência, a *query* para leitura da tabela usa do caractere “?” para remeter ao valor da variável declarada no fluxo de dados anterior ao *step,* e especificada no campo “*Insert data from step*”. A opção para tornar esta função disponível é selecionar o item “*Replace variables in script*”. Estas configurações podem ser visualizadas na Figura 23.

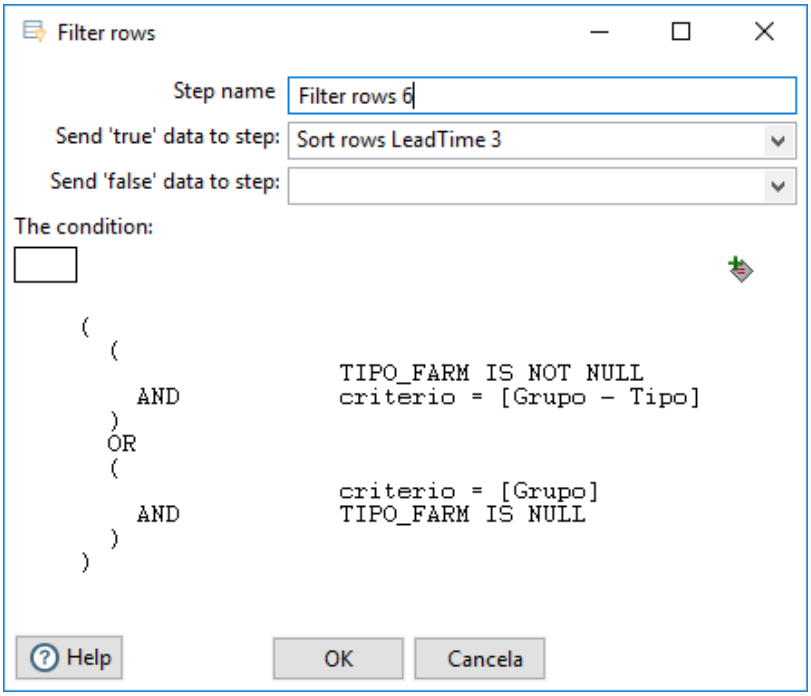
Figura 23 - *Step* de leitura de tabela usando variáveis no *script*.



Já para o cálculo por velocidade, uma diferente etapa utilizada é a de filtrar linhas, por exemplo, linhas nulas podem afetar *joins* futuros numa sequência lógica, tornando indesejável este tipo de dado. Para isso, pode-se utilizar o *step* "*Filter rows*", no qual se especifica o critério desejado para que os dados sigam no fluxo de dados programado.

A configuração deste bloco é apresentada na Figura 24. Neste exemplo é retratada uma condição específica do cálculo por velocidade de vendas, sendo que essa grandeza é relativa ao grupo ao qual a peça pertence ou a seu tipo, dependendo a escolha do usuário. Sendo assim, a escolha deste critério define se o campo "TIPO" será usado nos cálculos seguintes, nos quais valores nulos acarretam em erros, que devem ser negligenciados. Com isso, as tuplas que satisfazem a condição lógica descrita na janela "*The condition*" seguem no fluxo de dados do algoritmo.

Figura 24 - Aplicação *step Filter rows.*



Conforme citado, o algoritmo trabalha com diferentes partes denominadas *transformations.* Neste projeto, cada uma contém um tipo de distribuição ou funcionalidade especifica.

No caso do cálculo de distribuição por histórico, os inputs necessários e tabelas extraídas foram:

- Estoque disponível, Filiais: lidos de planilhas contidas em arquivos anteriormente extraídos.
- Necessidade de Atacado, Reguladores Preferenciais, Share de Vendas: extraídos com a execução de *queries* contidas na tupla presente no banco, proveniente da mensagem vinda da interface;
- ID: extraído do atributo contido na tabela proveniente da interface;

Já a tabela de *output* desta *transformation* contém os atributos com o valor real da distribuição, com a parte inteira deste valor e a parte fracionária, além dos demais campos com os dados das peças como cor, tamanhos, código identificador, entre outros.

Os dados resultantes da distribuição, com seus valores separados em diferentes campos, servem de entrada para o algoritmo Python, que como saída

tem os mesmos campos, porém com a parte inteira da distribuição como resultado final.

No cálculo por velocidade de vendas foram necessárias as entradas:

- Filiais, Tamanhos, Estoque: lidos de planilhas contidas em arquivos anteriormente extraídos.
- Lead Times, Filial e Regulador, Velocidade de Vendas, Grupo critério: extraídos com a execução de *queries* contidas na tupla proveniente da interface;

Como saída, a tabela contém dados referentes às peças por filial e um atributo com o valor a ser distribuído para cada granularidade.

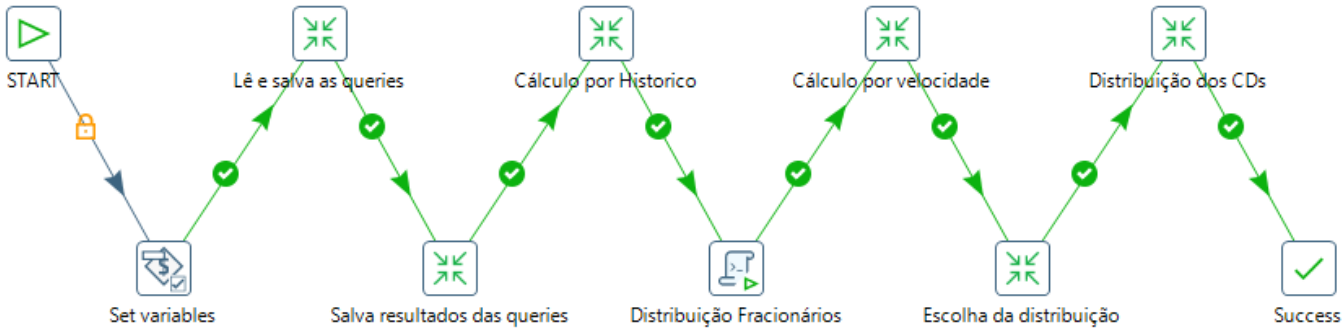
A *tranformation* responsável pela escolha entre as transformações utiliza dos *steps* e lógicas já apresentadas para definir qual resultado será o escolhido. Para isso, conta como entrada as duas tabelas resultantes das distribuições e como saída uma mesma tabela contendo os campos referentes a filial e peça, além do campo com o valor da distribuição final.

Vale ressaltar que os valores encontrados, por fim, são referentes a cada tamanho de modelo de peça por cada filial, tornando volumosa a tabela final.

4.3.4 MAIN JOB

Conforme apresentado no capítulo 4.1.1.1 Spoon, o Pentaho conta com *Jobs,* um tipo de aplicação que engloba e torna sequencial as diferentes *transformations* do algoritmo. Este é o bloco de execução principal que define o ponto de início e fim do algoritmo, além da ordem de execução das diferentes partes do cálculo da distribuição. Além disso, a *job* desenvolvida deve estar sempre disponível para ser requisitada pela API e executar o cálculo da distribuição. A *job* desenvolvida é apresentada na Figura 25.

Figura 25 - Main Job.



Cada um dos *steps* tem as seguintes funções:

- START: dá início ao processo.
- Set variables: atribui o id da distribuição em questão (recebido na chamada pela webAPI) à uma variável utilizada nas queries do algoritmo.
- Salva resultados das queries: como o nome sugere, executa as queries e salva as tabelas resultantes, ou seja, a extração das tabelas ESTOQUE, FILIAL e TAMANHOS.
- Cálculo por Histórico: executa o cálculo da distribuição por histórico, resultando num valor inteiro com parte fracionária.
- Distribuição Fracionários: este bloco faz a chamada do algoritmo em Python nomeado *algoritmo.py,* existente no mesmo diretório da *job*. Este algoritmo executa a distribuição das partes fracionárias dos valores resultantes da etapa anterior.
- Cálculo por velocidade: executa o cálculo da distribuição por velocidade.
- Escolha da distribuição: rotina que faz a escolha entre o valor final da distribuição por velocidade ou por histórico.
- Distribuição dos CDs: tendo a distribuição para as filiais realizada, este *step* executa o cálculo da distribuição para os Estoques Reguladores de RJ e SP.

4.4 Desenvolvimento da Web API

Conforme apresentado no Capítulo 2.3, a API tem como função tratar recursos para a utilização por outras aplicações, no caso deste projeto, executando a chamada do algoritmo e inserindo os dados necessários no banco.

Inicialmente, definiu-se o fluxo de informações e delegações que a própria API será responsável. Inicia-se com a requisição pela interface, que enviará a mensagem com os dados da distribuição. Em seguida, a API insere no banco de dados esses dados conforme o modelo de tabela criado para este fim. Ao inserir, lê-se o valor do atributo ID da tabela, identificador auto incremental e, consequentemente, único para cada distribuição. O valor deste atributo é retornado como mensagem de resposta da solicitação para a interface web para seu uso futuro. Na sequência, a API requisita o início do algoritmo passando o valor do ID como variável para sua execução. Como o algoritmo insere o resultado da distribuição com este mesmo identificador, a página web é capaz de ler este resultado em especifico.

Para facilitar o desenvolvimento desta aplicação, definiu-se a utilização do *framework* Express, que auxilia na integração com o banco de dados, juntamente com o *body-parser*, método para a leitura da mensagem que estará contida no corpo da mensagem POST.

A mensagem inicial da interface, dá-se através do método POST do protocolo HTTP, apresentado no Capitulo 2.3.1 Protocolo HTTP. Para a comunicação entre estes dois processos ocorrer com sucesso, definiu-se as configurações de porta do servidor hospedeiro e de URL, para que quando em produção, pudesse responder as solicitações conforme desejado.

Para essa troca de dados, buscou-se definir o modelo de mensagem para ser tratada. Posto que a linguagem escolhida para o desenvolvimento da API foi o JavaScript com interpretador Node.js, optou-se padronizar a maioria das informações nesta linguagem, fazendo assim a escolha pelo uso do JSON como formato de mensagem. A interpretação deste tipo de mensagem é simplificada pelo uso da biblioteca *body-parser* configurada no Express.

JSON é a sigla para JavaScript Object Notation, uma notação de troca de mensagens no formato de texto, que tem como principais vantagens a leveza

e a fácil interpretação humana. Sua simplicidade fez com que se popularizasse rapidamente, tornando-se tão usado quanto padrões já cimentados, como o XML.

O modelo de mensagem segue o padrão atributo-valor, conforme pode ser visto no exemplo de mensagem da Figura 26. Esta é uma amostra de mensagem utilizada na própria comunicação da ferramenta, a qual informações a respeito do cliente foram omitidas e seus valores foram substituídos pelo tipo de seu dado.

Figura 26 - Exemplo de mensagem JSON.

```
{
 "id_marca": <number>,
 "nome_distribuicao": "<string>",
 "data_criacao": <number>,
 "filial ": "<string>",
 "percentual": <number>,
 "reguladorl": <number>,
 " grupo _filiais": [
    {"data_criacao": "<date>",
     "id_grupo": <number>,
     "id_marca": <number>,
     "nome_ grupo ": "<string>",
     "tipo_ grupo ": "<string>",
     "ultima_alteracao": "<date>"
    },
    {"data_criacao": "<date>",
     "id_ grupo": <number>,
     "id_marca": <number>,
     "nome_ grupo": "<string>",
     "tipo_ grupo ": "<string>",
     "ultima_alteracao": "<date>"
    }],
 "query_estoque": "<string>”
}
```

Para interpretar os dados da mensagem, atribuíram-se estes a variáveis para posteriormente serem inseridos no banco. Todos estes dados serão utilizados em algum momento pelo algoritmo do Pentaho, que, com alguns destes, executará comandos em SQL para extração de dados do banco.

Para facilitar este processo, no algoritmo da própria API montou-se o comando em SQL com os dados recebidos e, em seguida, adicionou-se esta informação ao banco junto com os demais. O fato de ser executado na própria API facilita o processo para o algoritmo, já que a programação dos comandos no

formato de *strings,* em JavaScript, é menos custoso quando comparado ao mesmo processo no Pentaho.

Para a API inserir os dados recebidos na tabela do banco, criou-se a função *router,* que além de conter os parâmetros de conexão com o banco, recebe como parâmetro o tipo de operação desejado, como seleção (comando SELECT), inserção (INSERT), entre outros, e recebe também os dados a serem operados. O código e suas funções implementadas podem ser vistos no Apêndice A.

Como resposta do comando de inserção, também conhecido como *callback*, a API recebe o valor do identificador da tupla criada e o apresenta na mensagem de resposta do POST a interface.

Tendo este importante parâmetro, a API requisita a execução do algoritmo de ETL e cálculo da distribuição. Sua execução é realizada através da chamada *child_process,* biblioteca nativa do Node.js empregada quando há necessidade de utilizar comandos nativos do sistema operacional para a execução de algum processo. Com isso, o próprio sistema operacional irá gerenciar a execução do processo.

Nesta chamada, é passado o comando para a execução em *background* do algoritmo, o diretório da *job* e o parâmetro “id”. A compilação ocorre através do Kitchen, conforme apresentado no subcapitulo 4.1.1.2 Pan e Kitchen.

Por fim, a API retorna ao cliente uma mensagem pré-programada confirmando o funcionamento e a execução da ETL, juntamente da informação dinâmica do identificador da distribuição que está sendo calculada. Este identificador é usado pela interface para executar a leitura da tabela resultante do algoritmo, funcionalidade implementada pela equipe de desenvolvimento da empresa cliente.

Por ser *server-side,* a API deve sempre estar disponível para responder a solicitação do usuário através da interface web (*client*). Para isso, no servidor hospedeiro da API, utilizou-se do comando “*forever start*”, que faz com que a aplicação execute permanentemente.

Para testar o funcionamento da aplicação, fez-se uso do software Postman, que simula uma requisição do cliente ao servidor (API). Neste é

possível configurar um método completo, incluindo a mensagem e seu formato, e obter resposta do servidor.

Com isso, ao ser submetida para produção no servidor da empresa cliente, esta aplicação comportou-se conforme o esperado, tratando das requisições da interface web, transferindo dados do banco, e solicitando a execução do algoritmo de ETL.

5 RESULTADOS

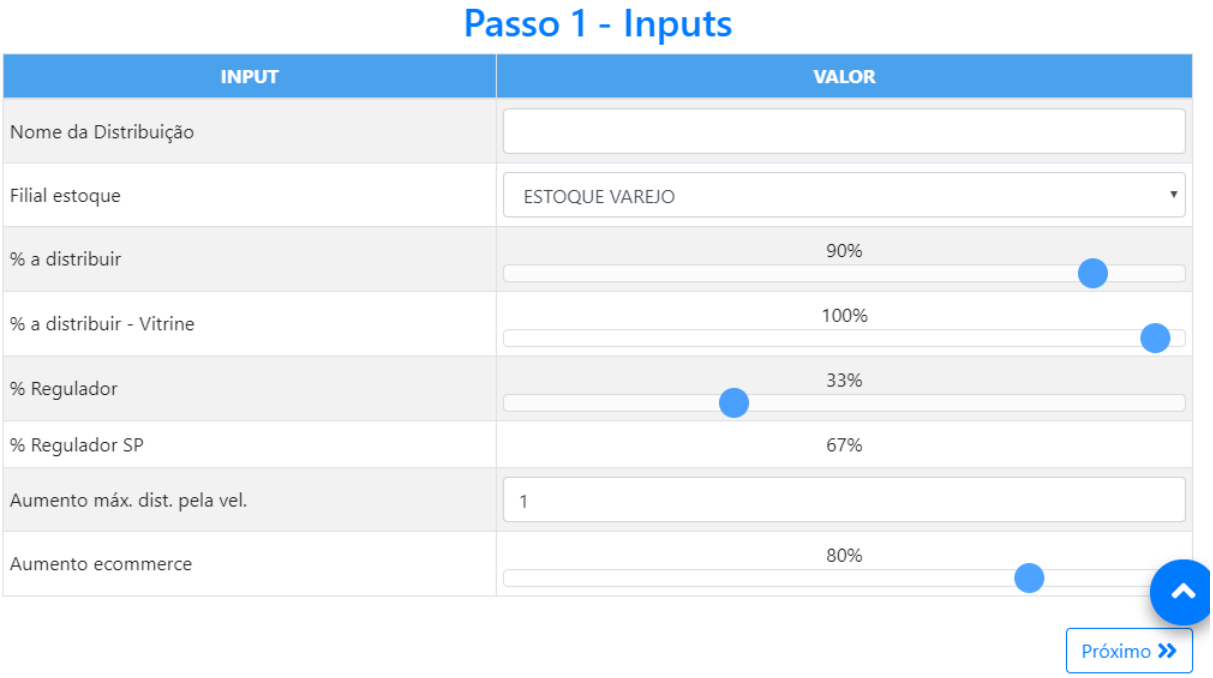
Como resultado da concepção deste projeto, obteve-se a implementação da ferramenta de distribuição de produtos, sendo utilizada no setor de estoque e logística da empresa cliente para o auxílio na tomada de decisão da quantidade de peças a se distribuir dos dois centros para as mais de 170 filiais do grupo de moda em todo o território nacional. Além disso, o projeto entregue entrou em atividade após um curto período de testes, contando atualmente com mais de 380 mil peças distribuídas por meio de seus cálculos.

Uma vez observados os principais requisitos inicialmente apresentados no Capítulo 1.2 Objetivos e o fato da ferramenta estar em uso junto do cliente, considera-se o desfecho do projeto bem-sucedido.

Uma destas necessidades a sanar com o projeto, foi a eficiência no uso, em outras palavras, tornar o processo mais automatizado e de fácil utilização. Isto foi possível com a abstração de processos manuais intermediários, antes executados pelo operador, através do desenvolvimento de aplicações voltados a comunicação entre seus processos, caso da API. Além disso, pode-se contar com o desenvolvimento de uma interface web que apresenta as informações de forma centralizada, com fácil inserção e edição dos dados.

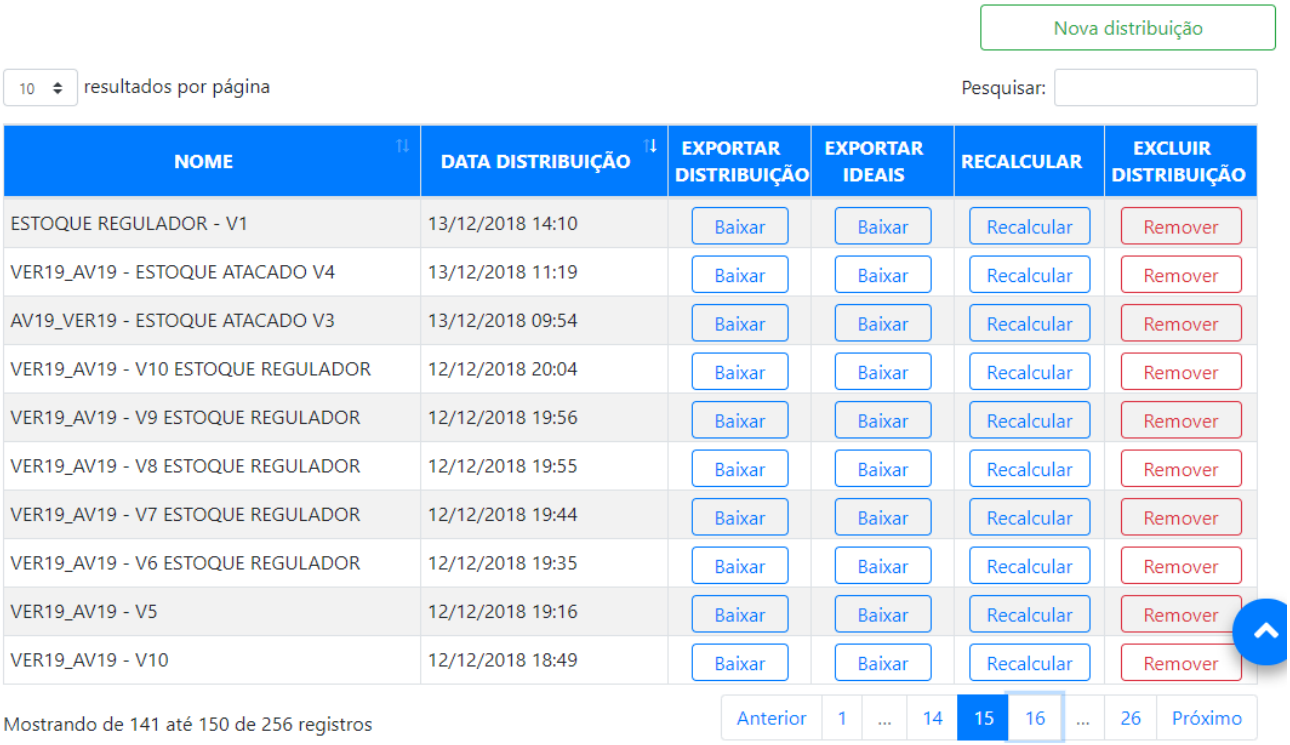
Na interface, que também se apresenta funcional no uso em *mobile*, o usuário inicialmente adiciona os dados da distribuição e, em seguida, submete estes ao cálculo. A janela apresentada é exibida na Figura 27.

Figura 27 - Interface inserção dos parâmetros de distribuição.



Enquanto isso, a ferramenta permite visualizar o histórico de distribuições, informando também o *status* da distribuição que está sendo executada. Nesta janela, é possível baixar o relatório da distribuição em forma de arquivo de tabela. Estas informações podem ser vistas na imagem a seguir (Figura 28).

Figura 28 - Janela de histórico de distribuições.



Quando concluída, o usuário é guiado para a janela dos resultados da distribuição, na qual é possível visualizar a quantidade de cada peça e tamanho destinados a cada filial. Caso necessário, pode editar e submeter novos valores, atualizando estas informações no banco de dados, desde que os novos valores estejam dentro das quantidades disponíveis no estoque. Nesta também é possível realizar o download da planilha gerada. Esta janela é visualizada na Figura 29.

Figura 29 - Janela de resultado da distribuição.

Como todas as etapas do processo passaram a ser executadas por aplicações voltadas, especificamente, a tratativas de grandes volumes de dados e sem intervenção humana entre estes, a confiabilidade e consistência dos dados é garantida.

Outro grande contratempo exposto foi o elevado tempo dispendido pelo analista na execução do procedimento de distribuição anteriormente utilizado. Com isso, buscou-se em todo processo empregar recursos que tornassem ágeis os cálculos e as etapas de ETL.

Considerando esta premissa, a ferramenta concebida apresenta desempenho satisfatório, visto que, segundo informações do cliente, o processo anterior consumia de uma a duas horas somente no processamento dos dados, ou seja, na execução do software. Além deste tempo, todo o processo contava com a mão de obra do analista que deveria realizar o *setup* inicial, com downloads e atualizações das planilhas e arquivos utilizados pelo software. Sendo assim, com a automação destas etapas e o eficiente cálculo, o tempo total despendido pelo processo e pela ferramenta, quando submetida aos testes, variou em torno de 30 a 45 minutos, ainda distante do tempo mínimo atingido pela ferramenta usada anteriormente.

Parametrizar um tempo fixo de execução da ferramenta é praticamente inviável, na medida em que o principal gargalo da variação de sua performance são as extrações provenientes das queries nos bancos de dados. Isto ocorre devido ao seu desempenho variar com a quantidade de dados a serem lidos e do momento que o banco está sendo requisitado, visto que o alto trafego de operações que ocorrem paralelamente acaba comprometendo seu desempenho. Apesar disso, os testes realizados demonstraram, em sua maioria, tempos de execução inferior a metade do tempo mínimo despendido no processo anterior.

Subtraindo este tempo do período de ociosidade do profissional de estoque e logística, responsável pela distribuição, e somando todos os processos de distribuição que este realiza, o ganho de produtividade que o uso da ferramenta resulta para a empresa é consideravelmente significativo. Em virtude do cálculo mais ágil, pode-se considerar que o funcionário terá ganho em torno de metade do tempo antes despendido no processo de ordenação do despache das peças para cada filial.

Os usuários do sistema, contam neste momento com uma ferramenta de fácil uso, com etapas sequenciais bem definidas, fundamentada em cálculos precisos e atualizados quanto a base central de dados. Tal sincronia, é possibilitada pela web API *server-side* desenvolvida, que trata as requisições de maneira eficiente no servidor, trafegando as informações necessárias para o correto fluxo do sistema.

Este ganho reflete até mesmo na própria logística de distribuição, tendo em vista que se pode organizar com maior velocidade um maior número de distribuições contando com dados provenientes de uma base atualizada, diminuindo a chance de falhas ou gargalos neste processo de planejamento.

Neste cenário, o cumprimento dos requisitos citados faz com que o uso da ferramenta entregue culmine na maior precisão do planejamento da distribuição de estoque, evitando erros de dimensionamentos de entregas e, consequentemente, gastos com retrabalho. Com isso, o correto uso da ferramenta proposta, ecoa em uma melhoria de todo o processo de estoque e logística. Conforme citado no início desse documento, toda etapa envolvida no *supply chain*, da recepção da matéria prima, até a entrega do produto a filial, tem

reflexo na satisfação do cliente e na manutenção da empresa no competitivo mercado.

Além disso, evita-se o custo de oportunidade, já que, com um dimensionamento equivocado da distribuição, o produto sendo transportado em vão poderia estar gerando lucro se alocado em outra filial.

Outro importante resultado é a viabilização de aperfeiçoamentos na tecnologia implementada, de modo que, acompanhando a entrega do software em produção, entregou-se também a documentação das etapas envolvidas nos cálculos das distribuições que, até aquele momento, era tido como nebuloso. Juntamente deste, apresenta-se o esclarecimento da lógica utilizada em cada etapa das aplicações, conforme requisitado pelo cliente. Com este material em mãos, novas versões de cálculos podem ser desenvolvidas, aperfeiçoando a lógica já presente na ferramenta. Todo este trabalho, só se tornou possível com a etapa inicial de concepção deste projeto e a explanação da lógica até então implementada.

Além da empresa cliente, a empresa concedente do estágio ao qual este projeto foi desenvolvido tem como um dos ganhos o valor da venda deste produto somado a sua renda. Ademais, durante todo o contato no processo de concepção da ferramenta, houve o estreitamento das relações com o cliente, promovendo oportunidades de futuros negócios.

6 CONCLUSÃO

Este projeto, desenvolvido junta à empresa Bix Tecnologia, resultou em um produto voltado para uma grande rede de lojas do setor da moda. Esse já vem sendo utilizado auxiliando o profissional da área de estoque e logística despachar a quantidade correta para a venda.

Através do simples uso da interface hospedada na página web, mais de um funcionário pode, simultaneamente, planejar e levantar dados precisos para novas distribuições de produtos, com maior velocidade e dados fidedignos quando comparado a ferramenta antes utilizada.

Além disso, fez parte do projeto concebido a documentação dos algoritmos implementados juntamente da explicação de seus cálculos, permitindo a empresa autonomia em desenvolvimentos e atualizações futuras.

Desse modo, o projeto possibilitou ampliar o limiar já abrangente da graduação em Engenharia de Controle e Automação, além de aplicar as aptidões desenvolvidas no decorrer do curso. Visto que a área de tecnologia e computação, a qual o curso está imerso, está em constante avanço, o profissional da área deve ser versátil e responder ao mercado para manter-se no meio de atuação.

Com isso, áreas de estudo recentemente incluídas no currículo do curso, como o *Data Mining* e *Data Warehouse*, além de disciplinas já cimentadas como Banco de Dados e Desenvolvimento de Sistemas, podem ser aplicadas com afinco no desenvolvimento deste projeto.

Ressalta-se, também, o trabalho em grupo realizado para a concepção da ferramenta em sua totalidade. Mesmo a longa distância, os profissionais envolvidos da empresa cliente desenvolveram a interface e prestaram as informações necessárias para o desenvolvimento dos algoritmos. Sem esta cooperação, o projeto não teria atingido os resultados esperados.

Assim, evidencia-se a importância das empresas em aliar a tecnologia em seu processo de cadeia de negócios, a fim de manter-se competitiva no mercado, sempre ressaltando a importância de seus dados e de sua análise. Sem eles, conclusões e resultados específicos, como os atingidos com a ferramenta, não seriam possíveis.

6.1 Sugestões para trabalhos futuros

Uma das imperfeições encontradas na lógica da aplicação foi no princípio de escolha entre as distribuições calculadas, por histórico ou velocidade de venda. Durante a implementação notou-se que havia inconsistências em seu método, e que poderiam ser aperfeiçoadas. Esta fora discutida com a equipe cliente a qual ficou ciente da possibilidade de melhorias.

A verificação da conclusão do término dos cálculos fica a compromisso da interface web, que periodicamente executa uma busca no banco pelo identificador da distribuição. Este processo não é tão otimizado, pois esta busca se repete algumas vezes até obter sucesso, ou seja, até encontrar o algoritmo ter terminado e o resultado estar presente no banco. Uma solução mais elegante seria a implementação de uma nova função na API para monitorar a conclusão do algoritmo de cálculo, e então notificar a interface para executar a busca. Apesar disso, o atual funcionamento não deprecia significativamente o atual funcionamento do sistema.

Outra possível melhoria pode ser implementada no que diz respeito a segurança da API. Apesar do nível de risco que expõe à empresa ser baixo, é sempre prudente evitar este tipo de contratempo. Para isso, pode-se aplicar técnicas como *Passport* ou *Tokens,* permitindo acesso a API somente requisições autenticadas e autorizadas.

Um dos resultados que podem ser levantados com o uso da ferramenta a médio prazo, é a assertividade dos cálculos, ou seja, se o resultado proveniente do sistema precisa, ou não, do ajuste fino do usuário. Com isso, pode-se obter informações importantes para futuras melhorias no sistema.

Com estas implementações, tem-se um valor ainda mais refinado para a distribuição, além da garantia do acesso as informações somente do usuário desejado, neste caso, o funcionário recorrendo da interface.

7 BIBLIOGRAFIA

BACKER, M. Spoon User Guide. **Pentaho Community Wiki**, 2015. Disponivel em: <https://wiki.pentaho.com/display/EAI/Spoon+User+Guide>. Acesso em: 11 dez. 2018.

BEZERRA, E. **Princípios de Análise e Projeto de Sistemas com UML**. 3ª. ed. Rio de Janeiro: Elsevier , 2015.

CESARIO, D.; COSTA, M.; DE SALLES, F. Pentaho Data Integration - ETL em Software Livre. **InfoQ Brasil**, 14 Novembro 2017. Disponivel em: <https://www.infoq.com/br/articles/pentaho-pdi>. Acesso em: 29 dez. 2018.

CÉSARO, O. **UTILIZAÇÃO DE FERRAMENTAS DE INFORMÁTICA DE REFINAMENTO E SIMULAÇÃO COM MODELOS MATEMÁTICOS NA GESTÃO DE ESTOQUES**. Dissertação (Dissertação de mestrado em administração)-UDESC. Florianópolis, p. 88. 2007.

CHANDRAYAN,. All About Node.Js. **codeburst.io**, 29 Outubro 2017. Disponivel em: <https://codeburst.io/all-about-node-js-you-wanted-to-know-25f3374e0be7>. Acesso em: 03 jan. 2019.

ENGEL, G. I. Pesquisa-ação. **Educar em Revista**, Curitiba , n. 16, p. 181-191, Dezembro 2000.

FRANCO, M. **Sistemas de Gerenciamento de Banco de Dados**. São João da Boa Vista: Instituto Federal de Educação, Ciência e Tecnologiade São Paulo, 2013.

GARCIA, E. S. et al. **Gestão de Estoques:** Otimizando a logística e a cadeia de suprimentos. Rio de Janeiro: E-papers, 2006.

HENRY, S. et al. **Engineering Trade Study: Estract, Transform, Load Tools for Data Migration**. University of Virginia. Charlottesville. 2005.

HITACHI VANTARA CORPORATION. Pentaho Data Integration. **Hitachi Vantara**, 2018. Disponivel em: <https://www.hitachivantara.com/en-us/products/big-data-integration-analytics/pentaho-data-integration.html?source=pentaho-redirect>. Acesso em: 23 nov. 2018.

KIMBALL, R.; ROSS, M. **The Data Warehouse Toolkit**. 3ª. ed. Indianapolis: John Wiley & Sons, 2013.

KUROSE, J.; ROSS, K. **Redes de Computadores e a Internet**. 5ª. ed. [S.l.]: Pearson, 2010.

LENZERINI,. **Data Integration:** A Theoretical Perspective. Conference: Proceedings of the Twenty-first. Wisconsin: [s.n.]. 2002. p. 233-246.

MILANI,. **MySQL - Guia do Programador**. [S.l.]: Novatec, 2007.

NAZÁRIO, P. A Importância de Sistemas de Informação para a Competitividade Logística. Disponivel em: <http://www.tecspace.com.br/paginas/aula/faccamp/TI/Texto04.pdf>. Acesso em: 08 Novembro 2018.

NODE.JS FOUNDATION. Express. **Express**, 2017. Disponivel em: <https://expressjs.com>. Acesso em: 21 dez. 2018.

NODE.JS FOUNDATION. About Node.js. **nodejs**. Disponivel em: <https://nodejs.org/en/about/>. Acesso em: 18 dez. 2018.

NODEBR COMMUNITY. O que é Node.js? **NodeBR**, 2016. Disponivel em: <http://nodebr.com/o-que-e-node-js/>. Acesso em: 18 dez. 2018.

ORENSTEIN, D. Application Programming Interface. **COPUTER WORLD**, 2000. Disponivel em: <https://www.computerworld.com/article/2593623/app-development/application-programming-interface.html>. Acesso em: 05 dez. 2018.

RODRIGUES MACHADO, ; ABREU, P. **Projeto de Banco de Dados:** Uma Visão Prática. 17. ed. [S.l.]: Saraiva Educação S.A., 2009.

ROTH, I. What Makes Node.js Faster Than Java? **StrongLoop by IBM**, 30 jan. 2014. Disponivel em: <https://strongloop.com/strongblog/node-js-is-faster-than-java/>. Acesso em: 03 jan. 2019.

SILBERSCHATZ , ; KORTH, H.; SUDARSHAN,. **Sistema de banco de dados**. 6ª. ed. [S.l.]: Elsevier Editora , 2006.

TANENBAUM, A. **Computer Networks**. 4ª. ed. [S.l.]: Prentice Hall, 2002.

TURBAN, E.; RAINER, K.; POTTER, R. **Administração de Tecnologia da Informação - Teoria e Prática**. 2ª. ed. São Paulo: Campus, 2003.

APÊNDICES

No apêndice são apresentados os algoritmos textuais implementados neste projeto. Inicialmente, tem-se o utilizado na concepção do cálculo e ETL, seguido do algoritmo do API *server*.

Apêndice A

Este algoritmo, descrito na linguagem Python, executa a distribuição da parte fracionária do valor encontrado no cálculo por histórico, dado que o resultado final deve ser um inteiro.

Inicialmente, tem-se a importação de bibliotecas para manipulação de tabelas e de cálculos, *pandas* e *math,* respectivamente. Em seguida, tem-se a importação do arquivo ".csv" contendo o resultado da distribuição. Na sequência, inicia-se o laço para a distribuição dos fracionários.

Posteriormente, para todos os tamanhos, um de cada vez, é realizada a soma dos valores inteiros (coluna INT_ADIST) e o total é subtraído da soma do resultado final (ADIST). Esta diferença é usada para ditar a quantidade de vezes que a distribuição dos fracionários acontecerá. Esta distribuição se dá varrendo todas as linhas (cada uma representando uma filial) da coluna de fracionários (FRAC_ADIST) buscando o maior valor, ou seja, a maior parcela fracionária. Encontrado este valor, subtrai-se um inteiro deste, com o objetivo de não ser pego pela próxima varredura, e soma-se um inteiro na coluna com os valores inteiros desta mesma linha. Ao fim, as filiais com maior parcela fracionária terão seus valores arredondados para cima, até que a soma do valor inteiro da distribuição esteja igual ao resultante a distribuir.

```
import pandas as pd
import math
df = pd.read_csv('FATO_DIST_HIST.csv',  encoding = "ISO-8859-1", sep=";", decimal=",")
```

```
        for i in range(6):
            coluna_inteira = 'INT_ADIST' + str(i+1)
            coluna_total = 'TOT_ADIST' + str(i+1)
            coluna_frac = 'FRAC_ADIST' + str(i+1)
            coluna_final = 'TOTFINAL_ADIST' + str(i+1)
            df[coluna_total] =  df[coluna_frac] +  df[coluna_inteira]
            df[coluna_final] =  df[coluna_inteira]
            soma_inteira = df[coluna_inteira].sum()
            soma_total = df[coluna_total].sum()
            diferenca = math.ceil(soma_total - soma_inteira)
            print(diferenca)
            for k in range(diferenca):
                df.loc[ (df[coluna_frac] == df[coluna_frac].max()),
coluna_final].iloc[0] = df.loc[ (df[coluna_frac] == df[coluna_frac].max()),
coluna_inteira].iloc[0] +1
                df.loc[ (df[coluna_frac] == df[coluna_frac].max()), coluna_frac].iloc[0]
= -1

        df.to_csv('FATO_DIST_HIST_FINAL2.csv', sep=';', encoding='utf-8')
```

Apêndice B

O algoritmo abaixo dita o funcionamento da API Node.js hospedada no servidor do cliente, que recebe e trata as solicitações da interface. Para sua concepção, utilizou-se da linguagem JavaScript, na qual linhas que iniciam com duas barras “//” não são compiladas, apresentando comentários sobre o código e complementando esta explanação.

Nas primeiras linhas, tem-se a adição das dependências utilizadas, caso do Express e do bodyParser, além da porta do servidor utilizada para troca de mensagens e das constantes locais utilizados no decorrer do código.

Trechos de código com informações relativas ao cliente foram omitidas.

```
const express = require('express');
var cors = require('cors');
const app = express();
const bodyParser = require('body-parser');
const port = <number>; //porta padrao
const mysql = require('mysql');

//configuração do body parser para método POST
app.use(bodyParser.urlencoded({extend: true}));
app.use(bodyParser.json());

//definição do roteador para requisição método GET para testes
const router = express.Router();
router.get('/', (req,res) => res.json({message: 'router/get - Funcionando'}));
app.use(cors());
app.use('/', router);

app.use(bodyParser.urlencoded({extended: true}));    //permite receber posts nos
formatos url encoded e json
app.use(bodyParser.json());

//rotina para adição de dados via POST
router.post('/distribuicao', (req, res) =>{

    const id_marca = req.body.id_marca;
```

```
...
const query_estoque = req.body.query_estoque;
const var_clusters_filial = req.body.clusters_filial;

//Laço para extração das filiais do campo clusters do json
for(i = 0; i < var_clusters_filial.length; i++){
   array = [var_clusters_filial[i].id_cluster];
   array.push(i);
}

const query_filial = ('SELECT codigo_filial as id_filial FROM Cluster_Filial where
id_cluster in ( '+ array+' )');
const clusters_filial = array.join(", ");

var sql = 'INSERT INTO Distribuicao SET ?';

const connection = mysql.createConnection({
   host     : '<string>',
   port     : '<number>',
   user     : '<string>',
   password : '<string>',
   database : '<string>'
});

var campos = {
   aumento_ecomm : aumento_ecomm,
   aumento_vel : aumento_vel,
   clusters_filial : clusters_filial,
   ...
   id_marca : id_marca,
   data_criacao : data_criacao,
   nome_distribuicao : nome_distribuicao,
};

//resposta do POST
```

```
   connection.query( sql, campos, function(error, results, fields){
      if(error)
         res.json(error);
      else
         var resposta = '{"Message":"ETL em
andamento","succes":true,"id":'+results.insertId+'}';
         res.json(JSON.parse(resposta));


         //requisição do algoritmo Pentaho
         const exec = require('child_process').exec;
         var yourscript = exec('/home/pentaho/7.1spoon/kitchen.sh -
file=/home/pentaho/ferramenta-distribuicao/etl-ferramenta-de-
distribuicao/MAIN_JOB.kjb -param:id_distribuicao='+results.insertId+' -
level=basic',//('cd C:/Users/ Documents/data-integration/kitchen.bat -
file=C:/Users/Documents/FerramentaDeDistribuicao/Algoritmo/MAIN_JOB.kjb -
param:id_distribuicao='+results.insertId+' -level=basic',
            (error, stdout, stderr) => {
               console.log(`${stdout}`);
               console.log(`${stderr}`);
               if (error !== null) {
                  console.log(`exec error: ${error}`);
               }
            });


      connection.end();
      console.log('executou!');
      console.log('Distribuição ' + results.insertId);      //retorno do Post com ID da
inserçãoo no banco
   });
});
//inicia o servidor
app.listen(port);
console.log('API funcionando!');
```