Universidade Federal de Pernambuco
Cin - Centro de Informática

Pós graduação em Ciência da Computação

ETL4NoSQL: Um framework de ETL para BDs NoSQL

Carine Calixto Aguena

Dissertação de Mestrado

Recife
<DATA DA DEFESA>

Universidade Federal de Pernambuco
Cin - Centro de Informática

Carine Calixto Aguena

ETL4NoSQL: Um framework de ETL para BDs NoSQL

Trabalho apresentado ao Programa de Pós graduação em Ciência da Computação do Cin - Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: *Valéria Cesário Times*

Recife
<DATA DA DEFESA>

<DIGITE A DEDICATÒRIA AQUI>

Agradecimentos

<DIGITE OS AGRADECIMENTOS AQUI>

<*DIGITE AQUI A CITAÇÂO*>
—<AUTOR> (<NOTA>)

Resumo

<DIGITE O RESUMO AQUI>

Palavras-chave: <DIGITE AS PALAVRAS-CHAVE AQUI>

Abstract

Keywords: <DIGITE AS PALAVRAS-CHAVE AQUI>

Sumário

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1

Introdução

Este capítulo contextualiza os principais assuntos abordados neste trabalho, apresenta as motivações, os objetivos gerais e específicos da proposta desta pesquisa, bem como sua justificativa.

1.1 Contextualização

Os requisitos para ferramentas de software modernas têm mudado significativamente, especialmente com o aumento das aplicações Web. Este segmento de aplicações exige requisitos com alta escalabilidade e vazão, onde sistemas que utilizam um armazenamento com esquema relacional não conseguem atender satisfatoriamente. Em resposta a isso, novas abordagens de armazenamentos de dados utilizando o termo de NoSQL tornaram-se popular.

O termo NoSQL é constantemente interpretado como "*Not Only SQL*", cujo SQL refere-se a linguagem de manipulação de dados dos gerenciadores de armazenamento de dados relacionais (RDBMS - Relational Database Management System) - Structure Query Language. O grande propósito das abordagens NoSQL é oferecer alternativas onde os esquemas relacionais não funcionam bem. Esse termo abrange diferentes tipos de sistemas. Em geral, banco de dados NoSQL usam modelo de dados não-relacionais, com poucas definições de esquema e escala horizontal [Nas12].

Muitas empresas coletam e armazenam milhares de gigabytes de dados por dia, no qual a analise desses dados torna-se uma vantagem competitiva no mercado. Dessa forma, há uma grande necessidade de uma nova arquitetura de Data Warehouse que possa alcançar melhor escalabilidade e eficiência [LTP13].

Segundo a definição de [KR02], Data Warehouse (DW) é uma coleção de dados para o processo de gerenciamento de suporte à decisão orientado a assunto, integrado, variante no tempo e não volátil. Os dados de diferentes fontes de sistemas são processados em um Data Warehouse central através da Extração, Transformação e Carga (ETL) de maneira periódica.

O projeto de ETL consome cerca de 70% dos recursos de implantação de um DW, pois desenvolver esse projeto é crítico e custoso, tendo em vista que gerar dados incorretos pode acarretar em más decisões. Porém, pouca importância foi dada ao processo de ETL por um grande período de tempo pelo fato de ser visto somente como uma atividade de suporte aos projetos de DW. Apenas a partir do ano 2000, a comunidade acadêmica passou a dar mais importância ao tema [dS12].

As pesquisas passaram a apontar problemas como complexidade, longa curva de aprendizagem, notações proprietárias, custo e tempo de implantação das ferramentas atuais. Além disso, é impossível oferecer um pacote fechado com todas as possibilidades de transformações exigidas pelos processos de ETL. Para sanar essas dificuldades, propostas de modelagens conceituais e lógicas foram apresentadas e Notação de Modelagem para Processo de Negócio (BPMN) para ETL foram definidas, as quais aumentam o nível de abstração dos processos de ETL, e consequentemente, os tornam independentes da plataforma de implementação. Contudo, a implementação dos processos de ETL programaticamente a partir de uma linguagem de programação de propósito geral tem sido adotada por muitas empresas, porque isso evita as desvantagens da utilização de ferramentas, além de aumentar o nível de customização e integração dos processos de ETL com outros sistemas [dS12].

Tradicionalmente, o DW é implementado em uma base de dados relacional, onde o dado é armazenado nas tabelas fato e tabelas dimensões, na qual forma um esquema em estrela (colocar ref DW). Por isso, é comum que as ferramentas de ETL utilizadas no mercado atualmente só dêem suporte aos esquemas relacionais. Para dar suporte aos sistemas que necessitem utilizar

um esquema não relacional em DW, a proposta desse trabalho é especificar um framework programável, flexível e integrado para modelagem e execução de processos ETL em BDs NoSQL.

1.2 Motivação

O aumento do uso de Banco de Dados com esquemas não relacionais e a falta de uma ferramenta programável, flexível e integrada, independente de plataforma que dê suporte à extração, transformação e carga em Data Warehouses é a grande motivação deste trabalho.

As pesquisas sobre extração de dados em BDs NoSQL mostram que não há uma ferramenta que seja integrada para o uso de BDs NoSQL, as ferramentas existentes no mercado apenas oferecem a possibilidade para alguns SGBDs NoSQL, ficando a cargo da equipe de implantação do projeto de DW todo o trabalho de modelagem e programação ao se utilizar BDs NoSQL.

[dS12] aponta em sua pesquisa que muitas empresas evitam ferramentas de ETL disponíveis no mercado, e adotam o desenvolvimento dos processos a partir de uma linguagem de programação de propósito geral, pelo fato dessas ferramentas terem uma longa curva de aprendizagem e grande complexidade no seu uso.

Dessa forma, encontrar uma solução que seja programável, flexível e integrada para extração, transformação e carga dos dados em BDs NoSQL é a motivação deste trabalho.

1.3 Objetivos

Nesta seção serão apresentados os objetivos geral e específicos desta pesquisa.

1.3.1 Objetivo Geral

Especificar um framework programável, flexível e integrado para modelagem e execução de processos ETL em BDs NoSQL.

1.3.2 Objetivo Específico

Estender a proposta do framework para facilitar a carga de dados de dois sistemas de BD NoSQL distintos baseado no paradigma família de coluna em um DW relacional.

1.4 Justificativa

1.5 Organização do Trabalho

CAPÍTULO 2

Fundamentação Teórica

Neste capítulo são apresentados os conceitos relacionados ao desenvolvimento desta pesquisa.

Os conceitos de ETL e Data Warehouse (DW), bem como o termo NoSQL e os paradigmas de esquemas não relacionais mais utilizados pela comunidade acadêmica, o Famílias de Colunas, Orientados à Documentos, Chave-Valor e Baseado em Grafos.

Também são detalhadas as definições de modelagem conceitual e lógica para esquemas não relacionais.

2.1 ETL

2.2 Data Warehouse

2.3 Bancos de Dados NoSQL

Consistem em bancos de dados não relacionais projetados para gerenciar grandes volumes de dados e que disponibilizam estruturas e interfaces de acesso simples (Lima; Mello, 2015). Cada paradigma NoSQL possui um esquema de modelagem diferente, nos quais são divididas pela literatura em quatro categorias amplamente usadas: Chave-Valor, Orientado a Documentos, Famílias de Colunas e Baseado em Grafos ([Fowler, 2013], [Kaur; Rani, 2013]).

As principais características dos banco de dados NoSQL são:

-Distribuído:

- Escalabilidade Horizontal
- Construído para grande volume de dados
- BASE ao invés de ACID
- Modelo de dados não relacional
- Sem definições de esquema
- Não suporta SQL

[Nas12]

2.3.1 Banco de dados Orientados à Documentos

Banco de dados orientados a documentos são capazes de armazenar documentos como dado. Esses documentos podem ser em qualquer formato como XML (eXtensible Markup Language), YAML (Yet Another Markup Language), JSON (JavaScript Object Notation), entre outros. Os documentos são agrupados na forma de coleções, comparando com banco de dados relacional as coleções são como tabelas e os documentos como os registros. Porém, a diferença entre eles é que cada registro na tabela do banco relacional tem o mesmo número de campos, enquanto que nos documentos na coleção do banco de dados orientado a documentos podem ter campos completamente diferentes (Kaur; Rani, 2013).

Existem mais de 15 banco de dados orientados a documentos disponíveis e os mais utilizados são MongoDB, CouchDB e o RavenDB (Kaur; Rani, 2013).

2.3.2 Banco de dados Famílias de Colunas

Banco de dados baseados em Famílias de Colunas são desenvolvidos para abranger três áreas: número enorme de colunas, a natureza esparsa dos dados e frequentes mudanças no esquema. Os dados em Famílias de colunas são armazenados em colunas de forma contínua, enquanto que em bancos de dados relacionais as linhas é que são contínuas. Essa mudança faz com que operações como agregação, suporte para ad-hoc e consultas dinâmicas se tornem mais eficientes (Kaur; Rani, 2013).

A maioria dos bancos de dados baseados em Famílias de Colunas são também compatíveis

com o framework MapReduce, no qual acelera o processamento de enorme volume de dados pela distribuição do problema em um grande número de sistemas. Os bancos de dados de Família de Colunas open-source mais populares são Hypertable, HBase e Cassandra (Kaur; Rani, 2013).

2.3.3 Banco de dados Baseado em Grafos

Bancos de dados baseado em Grafos são como uma estrutura de rede contendo nós e arestas, onde as arestas interligam os nós representando a relação entre eles. Comparando com o modelo Entidade-Relacionamento, o nó corresponde à entidade, a propriedade do nó à um atributo, a relação entre as entidades ao relacionamento entre os nós. Nos bancos de dados relacionais as consultas requerem atributos de mais de uma tabela resultando numa operação de junção, por outro lado, bancos de dados baseado em Grafos são desenvolvidos para encontrar relações dentro de uma enorme quantidade de dados rapidamente, tendo em vista que não é preciso fazer junções, ao invés disso, ele fornece indexação livre de adjacência (Kaur; Rani, 2013).

2.3.4 Banco de dados Chave-Valor

Em Bancos de dados Chave-Valor os dados são organizados como uma associação de vetores de entrada consistindo em pares de chave-valor. Cada chave é única e é usada para recuperar os valores associados a ele. Esses bancos de dados podem ser visualizados como um banco de dados relacional contendo múltiplas linhas e apenas duas colunas: chave e valor. Buscas baseadas em chaves resultam num baixo tempo de execução, além disso, os valores podem ser qualquer coisa como objetos, hashes, entre outros (Kaur; Rani, 2013).

Os bancos de dados Chave-Valor mais populares são Riak, Voldemort e Redis (Kaur; Rani, 2013).

2.4 Projeto Conceitual, Lógico e Físico

Tradicionalmente um projeto de banco de dados é modelado em três fases denominadas conceitual, lógica e física. O projeto conceitual consiste em apresentar um esquema expressivo que modele os dados de um determinado domínio de informação, enquanto que o projeto lógico transforma um esquema conceitual em algo que se aproxima de um modelo de implementação física do banco de dados. Em projetos de banco de dados NoSQL, há poucos trabalhos que abordam uma metodologia para esquemas lógicos baseados em modelagens conceituais (Lima; Mello, 2015).

Dessa forma, esta seção visa aprofundar o tema a respeito de projeto conceitual e lógico em banco de dados NoSQL.

2.4.1 Modelo Conceitual NoSQL

Em bancos de dados relacionais, o modelo conceitual mais utilizado na literatura é o modelo ER (Entidade-Relacionamento) (Fowler, 2013). Contudo, bancos de dados NoSQL necessitam de um modelo conceitual que atenda às suas características.

O desenvolvimento de banco de dados para sistemas NoSQL é usualmente baseado nas melhores práticas, nas quais são especificamente relacionadas ao sistema desenvolvido, com nenhuma metodologia sistematizada (Bugiotti; Cabibbo; Atzeni, 2014). Por isso, Bugiotti et al (2014) desenvolveu uma abordagem baseada no NoAM (NoSQL Abstract Model). Esta abordagem observa que vários sistemas NoSQL compartilham de características de modelagem similares. Uma importante observação é que sistemas NoSQL oferecem operações de acesso aos dados de forma eficiente, atômica e escalável nas unidades de acesso aos dados em uma certa granularidade. Uma representação errada pode levar a incapacidade de garantir a atomicidade das operações importantes e o desempenho pode piorar dependendo magnitude da aplicação.

A metodologia de Bugiotti et al (2014) procede com a identificação dos agregados, onde cada agregado é um grupo de objetos relacionados que podem ser acessados e/ou manipulados juntos. Essa atividade é importante para suportar escalabilidade e consistência. O modelo conceitual desenvolvido por Bugiotti et al (2014) segue o modelo padrão do DDD (Domain-Driven Design), no qual é uma metodologia muito utilizada em orientação à objeto. Dessa forma, para o modelo conceitual NoSQL, é utilizado o diagrama de classe conceitual da UML, definindo as entidades, valores dos objetos e relacionamentos da aplicação.

2.4.2 Modelo Lógico NoSQL

O modelo de dados lógico dominante nas últimas décadas tem sido o modelo relacional (Fowler, 2013). Porém, para bancos de dados NoSQL a modelagem relacional não atende as características para representação lógica de seus dados. Para Fowler (2013), cada solução NoSQL possui um modelo diferente, os quais ele dividiu em quatro categorias amplamente usadas na literatura: chave-valor, documento, famílias de colunas e grafos.

Nesse mesmo contexto, Lima (2015) propôs a utilização de esquemas lógicos para NoSQL que utilizam o conceito de agregados. Ele justifica a escolha pelo fato de que a representação lógica baseada em agregados apoia os requisitos típicos dos bancos de dados NoSQL, oferecendo suporte à escalabilidade, consistência e desempenho. O conceito de agregados é um termo da área Domain-Driven Design (DDD), sendo uma coleção de objetos relacionados, aninhados, representando como uma única entidade (Lima;Mello, 2015).

Agregado é um padrão de domínio usado para definir a propriedade e fronteira do objeto. Ele é um grupo de objetos associados que são considerados como uma unidade em relação a alterações de dados. O Agregado é demarcado pela fronteira que separa os objetos de dentro para fora. Cada Agregado tem uma raiz. A raíz é uma Entidade, e ela é o único objeto que é acessível de fora do agregado. A raíz pode guardar referências para qualquer dos objetos agregados, e os outros objetos podem guardar referências uns dos outros, mas um objeto de fora só pode guardar referências do objeto raíz. Se houver outras Entidades dentro da fronteira, a identidade dessas entidades é local, fazendo sentido somente dentro do agregado (Domain-

Driven Design Quickly, 2006).

Fowler (2013), define um agregado como um conjunto de objetos relacionados que são tratados como uma unidade, mais precisamente, é uma unidade de manipulação de dados e gerenciamento de consistência. Ele afirma também que trabalhar com banco de dados orientados a agregados traz uma semântica mais clara, enfocando a unidade da interação com o armazenamento de dados. Contudo, o motivo mais importante para a utilização da modelagem orientada a agregados em bancos de dados NoSQL é que ela auxilia a execução em um cluster. Quando se opera em um cluster é necessário minimizar o número de nós a serem pesquisados na coleta de dados. Assim, ao incluir os agregados é possível dar a informação ao banco de dados sobre quais partes serão manipuladas juntas e no mesmo nó.

Dessa forma, a utilização de um modelo lógico NoSQL baseado em agregados se justifica pelo fato de que o conceito desse modelo possibilita o gerenciamento de consistência, a execução em cluster e uma semântica mais clara.

2.5 Frameworks

2.6 Trabalhos Correlatos

Esta seção aborda os trabalhos que são correlatos a esta pesquisa, bem como descreve como estes trabalhos diferem do realizado por esta pesquisa.

2.6.1 Experiência do Usuário

Uma abordagem apresentado por Tableau permite em sua ferramenta que o usuário possa adicionar scripts Hive que consequentemente são executados em um Hadoop cluster. A saída é então importada para a memória no Tableau. O usuário define quais dados do sistema de arquivo devem ser processados, e o quanto complexo os dados deverão ser comprimidos para o formato que a aplicação suporte. Utilizando o cluster de várias máquinas esse processamento permite que o Tableau obtenha resposta rápida para um grande volume de dados até mesmo quando não há nenhum mecanismo de busca disponível. Obviamente, esta abordagem exige que o usuário tenha um conhecimento avançado, e também que haja uma análise a respeito da estrutura na qual os dados estão armazenados antes da importação.

[Nas12]

2.6.2 Solução Spotfire

CAPÍTULO 3

O Framework ETL4NoSQL

Neste capítulo serão apresentados os conceitos do Framework ETL4NoSQL. Este consiste numa plataforma de software para desenvolvimento de sistemas ETL, cujo os dados de origem provêm de bases de dados não relacionais. Mais especificamente, bases de dados NoSQL que pertencem a um dos quatro paradigmas de NoSQL: Orientada a documentos, Família de Colunas, Chave-Valor e Baseada em Grafos.

O Framework oferece um ambiente integrado para modelar processos de ETL e implementar funcionalidades utilizando uma linguagem de programação independente de uma GUI (*Graphical User Interface* - Interface Gráfica do Usuário).

Para a especificação do Framework foram definidas as estruturas dos dados dos ambientes de origem, destino e área de processamento de dados e suas respectivas linguagens de manipulação, e também, as principais funcionalidades dos sistemas de ETL, chamados mecanismos de ETL. Para realizar os processos de ETL foi definido um controlador de operações que é capaz de se comunicar com os ambientes e os mecanismos de ETL.

Nas seções deste capítulo serão detalhados os requisitos, a arquitetura, os fluxos de dados e diagramas utilizados no desenvolvimento do Framework.

3.1 Requisitos do ETL4NoSQL

Para que seja possível a extração, transformação e carga dos dados armazenados em bancos de dados que utilizam um dos paradigmas de NoSQL é preciso que seja definido o esquema no qual os dados necessários estão armazenados. Dessa forma, algumas questões importantes são abordadas:

- Quanto os BDs NoSQL diferem dos BDs suportados pelas ferramentas de ETL?
- Como é possível oferecer suporte para ETL em BDs NoSQL?

As diferenças dos BDs NoSQL já foram abordadas nos capítulos de fundamentação teórica. Assim, fica explícito o problema de como é possível converter os modelos de dados NoSQL em modelos relacionais que possam ser lidos por qualquer ferramenta de ETL. [Nas12] define uma tabela como uma representação de uma coleção de instâncias de entidades comparáveis. Então, possibilitar suporte para armazenamentos de dados NoSQL é permitir extração e importação de instâncias de entidades comparáveis em tabelas relacionais. Porém, os problemas a serem resolvidos abordados por [Nas12] para atingir isso incluem:

1. Como permitir o usuário especificar as entidades comparáveis e seus atributos?
2. Como, e de onde, extrair exatamente?

A solução sugerida pelo autor é um esquema ser deduzido por meio de uma amostra do banco de dados. Este esquema pode, então, ser apresentado ao usuário que procede selecionando quais atributos importar do que foi apresentado - exatamente como no caso do RDBMS. É claro que também deverá ser possível ao usuário editar os esquemas apresentados tendo em vista que a amostra pode não ser perfeita. Essa abordagem geral é referida como dedução de esquema - *schema inference*.

Construir uma amostra com todo o banco de dados pode ser muito custoso, por isso é preferível que faça uma amostra com pequenas estruturas, como por exemplo, uma entidade inteira. É claro que muitas vezes não há esse tipo de estrutura, então é possível que a amostra seja colocada em clusters. Múltiplas entidades podem ser divididas cada uma em um cluster, e assim, é possível ter um esquema de entidade por cluster.

Dessa forma, dado o que extrair, consultar e recuperar dados de um RDBMS é direto por causa do SQL. Sistemas NoSQL geralmente tem diferentes interfaces que suportam diferentes tipos de consultas. Então, não há nenhuma sugestão de solução geral para isso. Ao invés disso, uma investigação de cada interface particular dos sistemas deve ser conduzida para resolver o problema [Nas12].

Para suprir o problema a respeito das várias interfaces a serem lidadas na extração de dados de bases NoSQL, foi sugerido criar um ambiente programável que oferecesse interfaces previamente selecionadas e permitisse a inserção de novas interfaces por meio de linguagem de programação.

3.1.1 Descrição Geral do Sistema

O ETL4NoSQL consiste em um framework programável, flexível e integrado para modelagem e execução de processos ETL em BDs NoSQL.

Separamos os requisitos em duas etapas, os requisitos para as funcionalidades de ETL e os requisitos para a usabilidade de BDs NoSQL. Baseado nesses requisitos foi possível identificar os padrões de análise do framework, e assim, definir sua arquitetura.

3.2 Padrões de Análise de NoSQL

Segundo a pesquisa de Nasholm (2012), os requisitos necessários para lidar com dados que são armazenados em banco de dados não relacionais devem suportar buscas sob demanda. Devido à característica de bancos de dados NoSQL lidarem com um grande volume de dados é importante que as buscas sejam feitas conforme a necessidade desses dados. Dessa forma, os requisitos para BDs NoSQL são baseados nessa premissa, os padrões definidos pelos requisitos levantados foram os padrões de importação e padrões de mapeamento.

3.2.1 Padrões de importação

Padrões de importação são requisitos para possibilitar a importação dos dados de bases NoSQL, e assim, permitir a execução de processos ETL.

3.2.1.1 Definição de ambiente e estrutura de dados

a) Contexto: É preciso definir quais as estruturas da base de dados NoSQL a ser importada, bem como a linguagem de manipulação dos dados da base para que seja possível a importação.

b) Problema: É necessário oferecer a linguagem de manipulação da base de dados a ser importada, os tipos de dados que serão utilizados e local onde os dados se encontram.

c) Estrutura: Define as configurações de importação, oferecendo a informação de onde o dado é localizado, que dados incluir no resultado, como importá-lo e outras fontes de parâmetros específicos.

d) Participantes: Apresenta os elementos para a importação.

- Ambiente: fonte de onde o dado é localizado.
- Consulta: define quais dados serão necessários.
- Repositório: Define os atributos do repositório de dados, bem como a forma de conexão com o ambiente, forma de manipulação dos dados e outras configurações que permitam a leitura e escrita.
- Amostra: Define quais colunas e tipos de dados que serão utilizados para cada ambiente.

e) Próximos padrões: Concluindo as definições de ambiente e estrutura de dados, a próxima etapa é estipular os mecanismos de importação.

3.2.1.2 Mecanismos de importação

a) Contexto: É necessário definir mecanismos para a importação dos dados da base NoSQL de origem para permitir o uso desses dados em ferramentas de ETL.

b) Problema: É preciso determinar quais mecanismos são fundamentais para a importação dos dados.

c) Estrutura: O mecanismo pode ser definido como busca Sob Demanda ou busca Exaustivo. Mecanismo de busca Sob Demanda significa que o dado da fonte de dados só é buscado quando explicitamente demandado, e que não é buscado nenhum outro dado que não foi demandado. O mecanismo de busca Sob Demanda é muito importante quando se trata de análises em Big Data, pois as análises podem ser conduzidas de cima para baixo onde o usuário começa com uma visão agregada de um conjunto de dados e pode explorar detalhes se desejado. A visão inicial do agregado pode originar vários processos de ETL (Nasholm, 2012). Mecanismo de busca Exaustiva a busca é feita em uma única vez. O único requisito necessário é a configuração de importação.

d) Participantes:

Tabela 3.1 Mecanismos básicos para Importação

Fase de Importação	Mecanismo Padrão
Definição da estrutura de dados	Conexão ao Banco de dados Definição da linguagem de manipulação de dados
Busca	Sob Demanda Exaustiva

e) Próximos padrões: Após a definição dos mecanismos de importação, o próximo passo é executar o mapeamento para a estrutura que será utilizada pelos processos de ETL.

3.2.2 Padrões de Mapeamento

Padrões de mapeamento são requisitos para executar o mapeamento dos dados importados das bases NoSQL, modelando-os de forma que possibilite a leitura e execução dos processos de ETL.

3.2.2.1 Execução do processo de mapeamento

a) Contexto: Para cada paradigma NoSQL, é necessário definir regras de mapeamento que transforme o modelo não relacional em uma estrutura que mapeie as relações permitindo a leitura e execução de processos de ETL.

b) Problema: Deve-se identificar a estrutura da base de dados importada e possibilitar o mapeamento dessa base para um esquema que permita a leitura e execução de processos de ETL.

c) Estrutura: O mapeamento é definido de acordo com os quatro paradigmas NoSQL: chave-valor, família de colunas, orientado a documentos e baseado em grafos.

d) Participantes:

Tabela 3.2 Regras de Mapeamento

Paradigma NoSQL	Regras
Chave-Valor	Um campo chave pode ser mapeada como tabela relacional Uma coluna da chave correspondente pode ser mapeada como coluna da tabela da chave O valor correspondente de uma coluna é mapeada como linha da tabela
Família de Colunas	Uma família pode ser mapeada como tabela Uma coluna da família de colunas pode ser mapeada para uma coluna da tabela
Orientado a Documentos	Um documento corresponde a uma linha da tabela Cada campo do documento pode ser coluna da tabela relacional
Baseado em Grafos	O vértice ou aresta pode ser mapeado para uma tabela relacional Cada atributo do vértice ou aresta pode ser mapeado para colunas

e) Próximos padrões: Com as regras de mapeamento definidas, o passo seguinte é definir metadados e os mecanismos de ETL padrões para os processos de ETL.

3.2.3 Padrões de Análise de ETL

Segundo Silva (2012), não há um consenso a respeito das funcionalidades básicas dos sistemas de ETL, porém é possível definir, baseado nos conceitos disponíveis na literatura, requisitos de operações e metadados.

Requisitos de metadados definem as estruturas que cada processo de ETL necessita, onde podem ser consideramos pontos de flexibilidade. Já os requisitos de operações não mudam para qualquer processo de ETL, sendo assim, pontos de estabilidade.

Os principais requisitos de metadados e operações baseadas nas definições de Silva (2012) e consideradas fundamentais para o desenvolvimento do framework deste trabalho são descritas a seguir.

3.2.4 Padrões de Metadados

Padrões de metadados são requisitos de preparação do ambiente e definição de variáveis para a execução de operações de ETL.

3.2.4.1 Definição dos ambientes e das estruturas de dados

Este padrão consiste na preparação para a execução de processos de ETL.

a) Contexto: É necessário ao menos três ambientes para a execução de processos de ETL: fonte, área de processamento e destino.

b) Problema: É preciso definir que tipo de ambiente e estrutura de dados estarão disponíveis e permitirão acesso pelo framework.

c) Estrutura: Expõe sugestões para resolver o problema de definição dos ambientes, de acordo com cada requisito da estrutura dos dados. As definições de ambiente do framework proposto, para a parte de ETL, define a estrutura para os principais repositórios não relacionais e relacionais na literatura.

d) Participantes: Apresenta os elementos do ambiente e estrutura de dados.

   1. Ambiente: Define se o ambiente é fonte, destino ou área de processamento.
   2. Repositório de dados: Define os atributos do repositório de dados, bem como a forma de conexão com o ambiente, forma de manipulação dos dados e outras configurações que permitam a leitura e escrita.
   3. Esquema de dados: Define as tabelas, campos, relacionamentos, visões, índices, pacotes, procedimentos, funções, filas, gatilhos, tipos, sequências, visões materializadas, sinônimos, enlaces de banco de dados, diretórios, esquemas XML e outros elementos dos ambientes.
   4. Amostra: Define quais colunas e tipos de dados que serão utilizados para cada ambiente.

e) Próximos padrões: Após as definições de ambientes e estrutura de dados, o passo seguinte é definir os mecanismos padrões para o ETL.

3.2.4.2 Mecanismos de ETL

a) Contexto: É necessário definir mecanismos básicos para a execução de processos de ETL. Os mecanismos mais comuns encontrados na literatura (Silva, 2012) estão expostos na tabela 3.1.

Figura 3.1 Diagrama de Classe da Definição dos Ambientes e Estrutura de dados

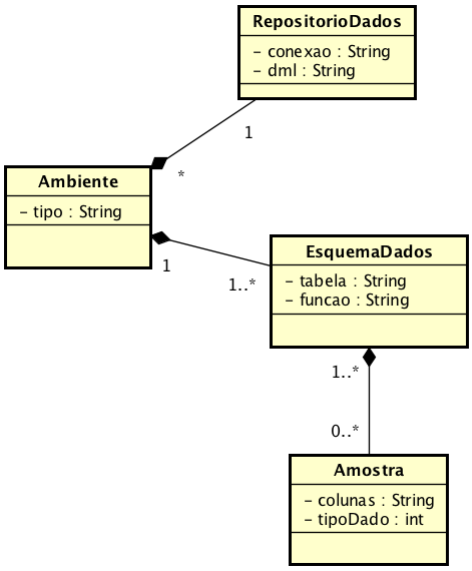


Tabela 3.3 Mecanismos básicos para o padrão de análise Mecanismos de ETL

Fase de ETL	Mecanismo Padrão
Extração	Extrair dados
Transformação	Filtrar dados Unir dados Agregar dados Juntar dados Gerar Chaves Pesquisar chaves Converter dados
Carga	Carregar dados

b) Problema: É necessário determinar quais os mecanismos fundamentais para o funcionamento dos recursos padrões de ETL. Dessa forma, determinar quais processos deverão estar disponíveis, e como utilizá-los.

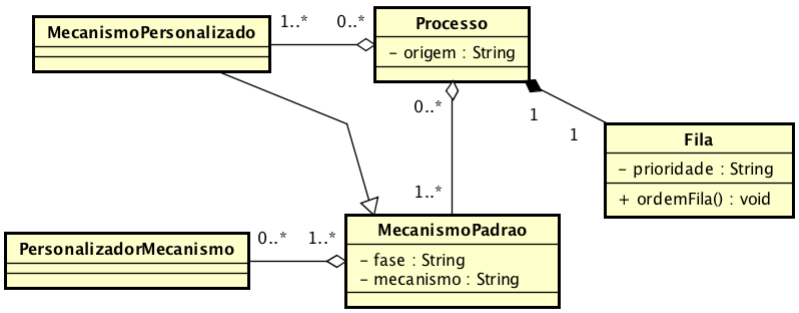
c) Estrutura: Define os mecanismos padrões utilizados pelos processos de ETL. Porém, esse conjunto de mecanismos pode não atender a necessidade de todas as aplicações, é necessário permitir a criação de mecanismos personalizados para um determinado domínio de aplicação. Além disso, deve ser possível personalizar e reutilizar os mecanismos existentes.

d) Participantes: Apresenta os elementos dos mecanismos comuns e personalizados.

1. Fila: conjunto de processos de um mecanismo em ordem de execução.
2. Processo: área de processamento, origem do processo, atributos do processo.

3. Mecanismo Padrão: mecanismos básicos mais relevantes para a execução de processos de ETL.
4. Mecanismo Personalizado: mecanismos personalizados para atender domínios específicos.
5. Personalizador de mecanismo: tem a função de permitir ao programador alterar comportamentos dos mecanismos padrões.

Figura 3.2 Diagrama de Classe Mecanismos de ETL



e) Próximos padrões: O passo seguinte à modelagem dos mecanismos padrões de ETL é a definição dos padrões de operação.

3.2.5 Padrões de Operação

Padrões de Operação são requisitos para a execução dos processos de ETL.

3.2.5.1 Leitura dos ambientes e das estrutura de dados

a) Contexto: É preciso executar a leitura dos ambientes e de estrutura de dados que farão parte dos processos de ETL. Para a execução dos mecanismos padrões é necessário que haja a definição e posteriormente a leitura dos dados.

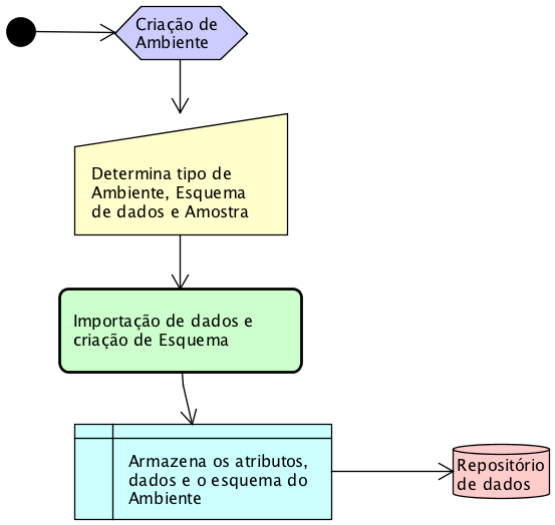
b) Problema: A leitura dos dados deve ser feita de forma que possibilite a execução dos processos de ETL por meio dos mecanismos padrões ou personalizados.

c) Estrutura: A operação de leitura é feita por meio da instanciação dos elementos:

1. Ambiente: determina o método de criação do ambiente, bem como a instanciação dos seus atributos.
2. Repositório de dados: determina o método de criação de repositório de dados, onde é possível definir quais tipos de dados estão presente no determinado ambiente.
3. Esquema de dados: determina o método de leitura do esquema dos dados do ambiente, essa leitura é feita por meio da linguagem de manipulação de dados.

4. Amostra: determina a leitura das colunas e tipos de dados de um determinado esquema de dados.

Figura 3.3 Fluxograma da criação de Ambientes



d) Participantes: apresentação os principais métodos para leitura dos ambientes e estrutura de dados.

Tabela 3.4 Métodos de Leitura de Metadados

Descrição	Método
Conexão ao Banco de dados	connectDB
Criação de Esquema de dados	createSchema
Armazena os dados	setData
Consulta os dados	getData
Criação de Amostra	createSample

e) Próximos padrões: a próxima etapa a ser definida é o padrão da execução dos processos de ETL.

3.2.5.2 Execução dos processos de ETL

a) Contexto: Por meio da definição dos mecanismos básicos dos processos de ETL é possível realizar a sua execução.

b) Problema: A execução dos processos deverá ser feita de acordo com a seu grau de prioridade definida pelo usuário.

c) Estrutura: A execução dos processos é feita por meio da definição do seu mecanismo.

d) Participantes: Mecanismos dos processos de ETL.

e) Próximos padrões: Após a execução dos processos de ETL, o passo seguinte é a definição dos padrões de análise de NoSQL para permitir o uso de banco de dados do paradigma NoSQL pelo framework.

Tabela 3.5 Padrões de Análise do ETL4NoSQL

Subsistema	Padrão de Análise	Etapa
ETL	Metadados	- Definição de Ambiente e Estrutura de dados - Mecanismos de ETL
ETL	Operação	- Leitura de Ambiente e Estrutura de dados - Execução dos mecanismos de ETL
NoSQL	Importação	- Definição de Ambiente e Estrutura de dados Mecanismos de NoSQL
NoSQL	Mapeamento	- Leitura de Ambiente e Estrutura de dados - Execução dos mecanismos NoSQL

Com isso, por meio das definições de requisitos foi possível definir os padrões de análise que estão apresentados na tabela 3.5.

3.3 Arquitetura do ETL4NoSQL

A arquitetura do Framework será apresentada nesta seção, ela foi definida baseada em componentes. Segundo [Sam97], componentes são uma parte do sistema de software que podem ser identificados e reutilizados, onde descrevem ou executam funções específicas e possuem interfaces claras, documentação apropriada e a possibilidade de reuso bem definida. Ainda de acordo com o autor, um componente deve ser autocontido, identificável, funcional, possuir uma interface, ser documentado e ter uma condição de reuso. Dessa forma, definimos os componentes do ETL4NoSQL de acordo com essas características e serão apresentados na seção 3.3.1.

3.3.1 Componentes do ETL4NoSQL

A seguir são apresentados os modelos de componentes do Framework ETL4NoSQL de acordo com os padrões de análise e as convenções estabelecidas por [Hei01].

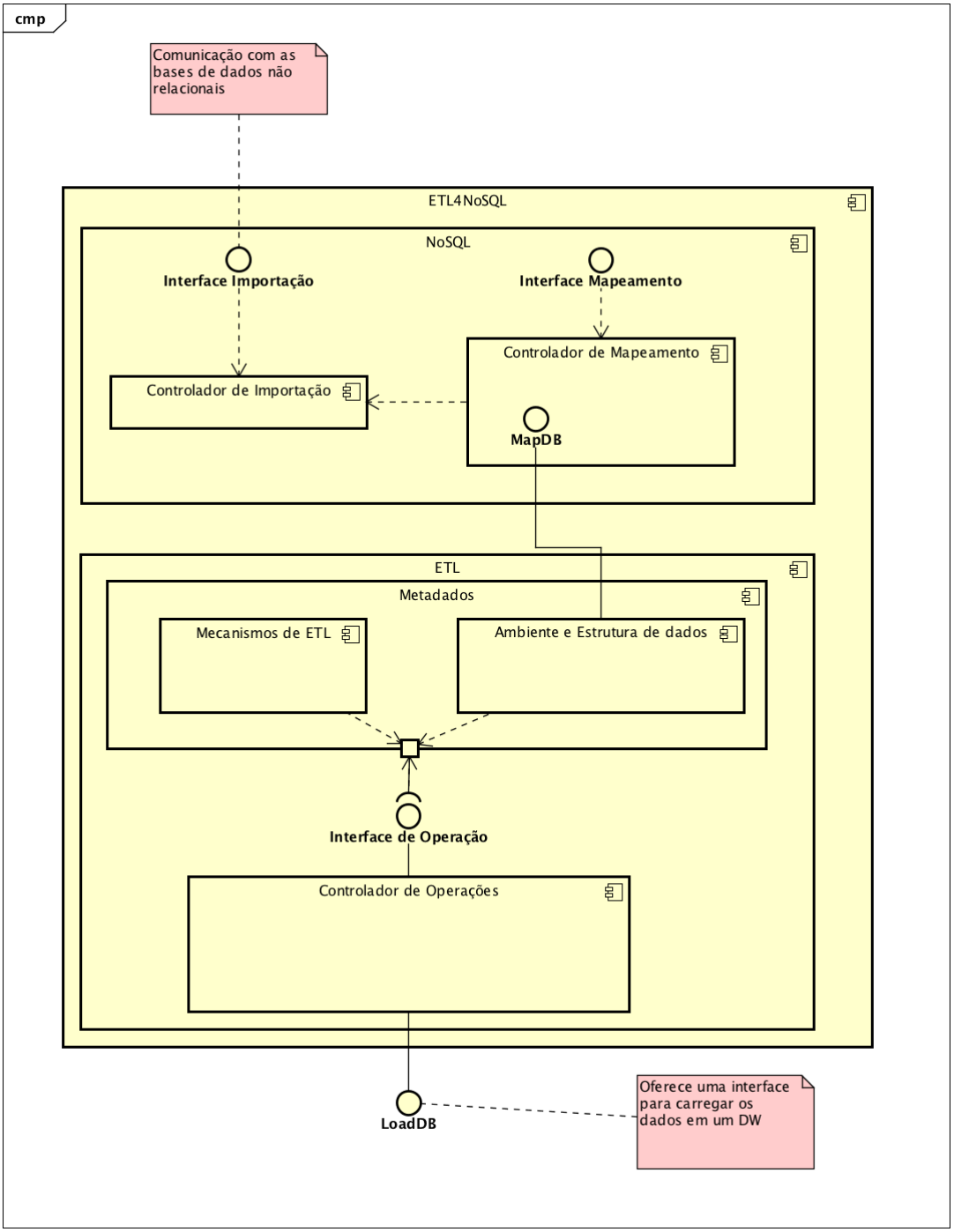


Figura 3.4 Arquitetura do Framework baseada em Componentes

3.3.1.1 Componente de Importação

a) Interface: Componente responsável pela importação dos dados da base de origem

b) Nomeação: Componente de Importação

c) Metadados: Este componente será composto com as informações da base de origem e da busca dos dados, apresentará uma interface para o usuário disponibilizar as informações e fornecerá os dados para o componente de mapeamento.

d) Interoperabilidade: Deve possibilitar a comunicação entre outros componentes

e) Customização: Este componente pode ser customizado em sua interface com o usuário, possibilitando outras meios de fornecer as informações para os seus métodos

f) Suporte a evolução: Deve possibilitar o suporte aos métodos de acordo com as mudanças de conexões e manipulações de bases de dados futuras

g) Empacotamento e utilização: Os métodos deverão ser encapsulados e poderão ser utilizados pela importação de sua classe e a interface com o usuário será por meio de linha de comando

3.3.1.2 Componente de Mapeamento

a) Interface: Componente responsável por gerar o mapeamento dos dados oferecidos pelo componente de importação para um esquema relacional

b) Nomeação: Componente de Mapeamento

c) Metadados: Este componente deverá estabelecer comunicação com o componente de importação, apresentará uma interface para o usuário definir a regra a ser utilizada no mapeamento e fornecerá os dados para o componente de leitura e escrita de metadados

d) Interoperabilidade: Deve possibilitar a comunicação entre outros componentes

f) Customização: É possível customizar as regras de mapeamento para outros esquemas não relacionais

g) Suporte a evolução: Deve possibilitar o suporte aos métodos de acordo com a necessidade de alterar os esquemas dos dados

h) Empacotamento e utilização: Os métodos deverão ser encapsulados e poderão ser utilizados pela importação de sua classe e a interface com o usuário será por meio de linha de comando

3.3.1.3 Componente de Leitura e Escrita de Metadados

a) Interface: Componente responsável pela criação de ambientes para a execução de processos de ETL, importação e mapeamento NoSQL.

b) Nomeação: Componente de Leitura e Escrita de Metadados

c) Metadados: Este componente deverá estabelecer comunicação com o componente de importação caso o ambiente seja fonte de dados, deve criar um esquema para o ambiente de processamento de dados, um esquema para o ambiente de destino e uma amostra para os dados importados

d) Interoperabilidade: Deve possibilitar a comunicação entre outros componentes

f) Customização: É possível customizar os ambientes de acordo com a necessidade do usuário

g) Suporte a evolução: Deve possibilitar o suporte aos métodos de acordo com a necessidade de alterar os esquemas dos dados

h) Empacotamento e utilização: Os métodos deverão ser encapsulados e poderão ser utilizados pela importação de sua classe e a interface com o usuário será por meio de linha de comando

3.3.1.4 Componente de Mecanismos de ETL

a) Interface: Componente responsável por estabelecer os mecanismos que estarão disponíveis para a execução de processos de ETL

b) Nomeação: Componente Mecanismos de ETL

c) Metadados: Este componente deverá estabelecer comunicação com o componente de Leitura e Escrita de Metadados, apresentará uma interface para o usuário definir quais mecanismos a serem utilizados, bem como manterá os mecanismos que estarão disponíveis para o uso

d) Interoperabilidade: Deve possibilitar a comunicação entre outros componentes

e) Composição:

f) Customização: É possível customizar e criar mecanismos de acordo com a necessidade de cada processo de ETL

g) Suporte a evolução: Deve possibilitar o suporte aos métodos de acordo com a necessidade de alterar os esquemas dos dados

h) Empacotamento e utilização: Os métodos deverão ser encapsulados e poderão ser utilizados pela importação de sua classe e a interface com o usuário será por meio de linha de comando

3.3.1.5 Componente de Operações

a) Interface: Componente responsável por criar e executar processos de ETL

b) Nomeação: Componente de Operação

c) Metadados: Este componente deverá possibilitar a comunicação com o componente de metadados e o componente de mecanismos de ETL e deverá criar e executar processos de ETL

d) Interoperabilidade: Deve possibilitar a comunicação entre outros componentes

f) Customização: É possível customizar os processos de ETL criados

g) Suporte a evolução: Deve possibilitar o suporte aos métodos de acordo com a necessidade de alterar os processos

h) Empacotamento e utilização: Os métodos deverão ser encapsulados e poderão ser utilizados pela importação de sua classe e a interface com o usuário será por meio de linha de comando

3.3.2 Definição de Inventário NoSQL

A partir da análise de padrões e definição dos componentes de Importação e Mapeamento foi possível projetar o modelo de interação entre esses componentes, mostrado na figura 3.5. Ele fornece uma visão conceitual da relação entre os componentes para a construção do inventário aplicado na importação dos dados das bases NoSQL e no mapeamento para o esquema relacional utilizado na fase de ETL.

O inventário NoSQL é composto pelas entidades:

- Inventário Metadados: é a entidade responsável por manter, adicionar e remover um conjunto de inventários;
- Esquema de dados: entidade que define o comportamento e os atributos de um esquema de dados, é possível derivar outras entidades de esquema específicos a partir dela;
- Inventário: entidade que mantêm a descrição do inventário, dados de conexão e o conjunto de esquemas que precisam estar disponíveis para a importação e mapeamento do dados NoSQL;
- Sintax DDL: entidade que define o comportamento e os atributos da linguagem de definição de dados de um SGBD;
- Sintax DML: entidade que define o comportamento e os atributos da linguagem de manipulação de dados de um SGBD.

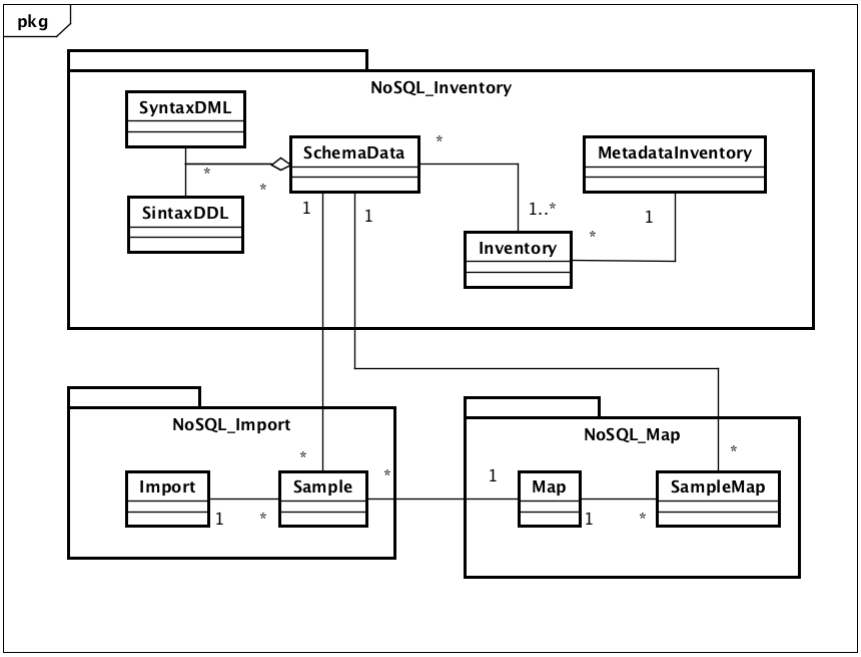


Figura 3.5 Diagrama de Classes Modelo de Interação entre Componentes

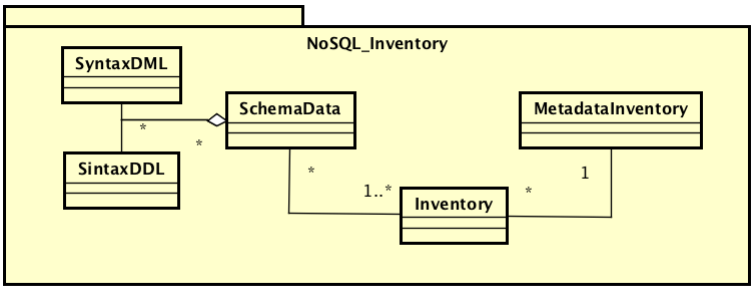


Figura 3.6 Diagrama de Classes do Componente Inventário

3.3.2.1 Instanciação do Inventário NoSQL

A construção de aplicações baseadas no ETL4NoSQL consiste na importação de dados de bases não relacionais seguido do mapeamento desses dados para uma forma relacional que possam fazer parte da criação de processos de ETL utilizando uma linguagem de programação de propósito geral. Para isso, a interface de programação para a instanciação do inventário é composta pela criação da classe MetadaInventory, onde é feita a chamada do método addInventory que adiciona o inventário por meio dos parâmetros schemaData, nameInventory, describeInventory.

3.4 Considerações finais

CAPÍTULO 4

Ambiente de Programação

Este capítulo consiste na apresentação do ambiente de programação do ETL4NoSQL. Ele foi desenvolvido utilizando a linguagem de programação orientada a objetos Python. É demonstrado também os aspectos de implementação, as classes de software e as instanciações dos objetos das classes. As classes são interfaces de programação orientada a objetos fundamentais para o modelo de abstração de frameworks, e as classes deste trabalho serão utilizadas para a importação, mapeamento de BDs NoSQL e criação de processos de ETL a partir desses BDs.

4.1 Implementação

A implementação do ETL4NoSQL foi feita utilizando a linguagem de programação orientada a objetos Python. A escolha dessa linguagem justifica-se pelo fato dela utilizar o paradigma de orientação a objetos que é adequada para a implementação dos padrões de projeto desenvolvidos na proposta deste trabalho. Além disso, Python tem uma sintaxe de fácil aprendizado e pode ser usada em diversas áreas, como Web e computação gráfica. Ela é uma linguagem de alto nível interpretada, completamente orientada a objetos e também é um software livre.

Assim, a implementação do framework foi baseada nos princípios do design orientado a objetos de inversão de controle, onde determina que os módulos de alto nível não devem ser dependentes de módulos de baixo nível, e sim, de abstrações, ou seja, os detalhes devem depender das abstrações. Esse princípio sugere que dois módulos não devem ser ligados diretamente, pois devem estar desacoplados com uma camada de abstração entre eles. Para suprir esse princípio, o ETL4NoSQL tem uma classe abstrata para o Esquema de Dados, que utiliza dos mesmos comportamentos, para as diversas variações de esquemas que os paradigmas NoSQL possui, porém aplicados de acordo com a especificidade de cada um. Outro princípio importante utilizado é o da segregação de interfaces onde os usuários não devem ser forçados a depender de interfaces que não necessitam, deve-se escrever interfaces enxutas com métodos que sejam específicos da interface.

Portanto, o framework foi dividido em interfaces de importação, mapeamento dos BDs NoSQL, mecanismos e operações de processos de ETL. As ferramentas utilizadas para implementação do ETL4NoSQL foram:

- Notebook com sistema operacional MacOS X; processador de 2,5 GHz Intel Core i5; e memória 12 GB 1333 MHz DDR3;
- Python 2.7: Linguagem de programação orientada a objetos.
  Disponível em: https://www.python.org/download/releases/2.7/;
- LiClipse 3.4.0: plataforma de programação (IDE) open-source.
  Disponível em: http://www.liclipse.com/download.html;
- SGBD MariaDB versão 10.0.27.
  Disponível em: https://downloads.mariadb.org/mariadb/10.0.27/;
- SGBD Redis versão 3.2. Disponível em: https://redis.io/download;
- SGBD Cassandra 3.0. Disponível em: http://cassandra.apache.org/download/

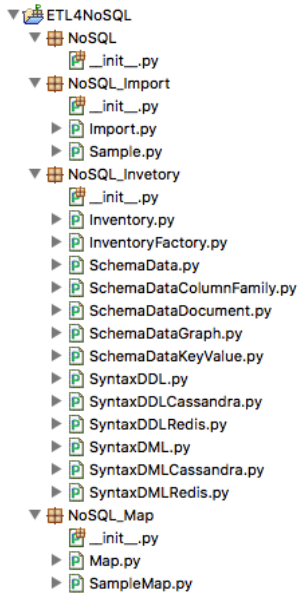
4.2 Interfaces de Programação

4.2.1 Módulo NoSQL

O módulo NoSQL é responsável por lidar com toda a parte que diz respeito ao dados modelados a partir dos paradigmas NoSQL. É neste módulo que serão feitas as importações dos

dados e mapeamentos dos esquemas das bases não relacionais. A árvore de organização das classes do módulo pode ser vista na Figura 4.1.

Figura 4.1 Árvore das Classes do Módulo NoSQL



4.2.1.1 Esquema de Dados - SchemaData

Esquema de dados é uma classe que utiliza o padrão factory method, esse padrão permite que as interfaces criem objetos, porém a responsabilidade de criação fica a cargo da subclasse, as classes que derivam dela são as variações de esquemas existentes dos vários paradigmas de armazenamento de dados presentes na literatura. O trecho do código está ilustrado na Figura 4.2.

O método createSchema é um método abstrato, e é por meio dele que as subclasses implementarão a criação dos seus esquemas de maneira personalizada.

4.2.1.2 Esquema de Dados Família de Coluna - SchemaDataColumnFamily

Classe do tipo ConcreteCreator derivada do Esquema de Dados onde define a criação do esquema das bases sob o paradigma NoSQL Família de coluna. Segundo Nasholm (2012), o esquema de dados do paradigma Família de Coluna é uma coleção de colunas em uma tabela. As linhas são similares às linhas do esquema relacional, exceto que todas as linhas na mesma tabela não necessariamente tem a mesma estrutura. Um valor numa célula, por exemplo, a intersecção de uma linha e uma coluna, é uma sequencia não interpretada de bit. Cada célula é versionada, significando que ela contém múltiplas versões do mesmo dado e que cada versão tem um timestamp atrelada a ela.O trecho do código está ilustrado na Figura 4.3.

Figura 4.2 Classe Schema Data

```
from abc import abstractmethod

class SchemaData:
    def __init__(self, name, describe):
        self.columns = []
        self.name = name
        self.describe = describe
        self.createSchema()

    @abstractmethod
    def createSchema(self):
        pass

    def getSchema(self):
        return "Name Schema: " + self.name + " Describe Schema: " + self.describe

    def putColumn(self, key):
        self.columns.append(key)

    def getColumn(self, key):
        index = self.columns.index(key)
        return self.columns[index]

    def getColumns(self):
        return self.columns

    def popColumn(self, key):
        self.columns.pop(self.columns.index(key))
```

Figura 4.3 Classe Schema Data Column Family

```
from NoSQL_Invetory.SchemaData import SchemaData


class SchemaDataColumnFamily(SchemaData):

    def createSchema(self):
        key = raw_input("Insert key value: ")
        timestamp = raw_input("Insert timestamp ")
        self.putColumn([key, timestamp])
```

4.2.1.3 Esquema de Dados Documento - SchemaDataDocument

Esquema de Dados Documento é uma subclasse do Esquema de Dados do tipo Concrete-Creator, ela define o esquema das bases sob o paradigma NoSQL Orientado a documento. Conforme Nasholm (2012), um documento é geralmente um conjunto de campos onde o campo é um par chave-valor. Chaves são strings atomicas ou sequencia de bits, e valores também são atomicos, por exemplo, inteiros ou strings, ou complexos, por exemplo, listas, mapas, entre outros. Um armazenamento de dados de documentos pode armazenar muitos documentos ou até mesmo muitas coleções de documentos. O trecho do código está ilustrado na Figura 4.4.

Figura 4.4 Classe Schema Data Document

```
from NoSQL_Invetory.SchemaData import SchemaData


class SchemaDataDocument(SchemaData):

    def createSchema(self):
        key = raw_input("Insert key value: ")
        document = raw_input("Insert document ")
        self.putColumn([key, document])
```

4.2.1.4 Esquema de Dados Grafo - SchemaDataGraph

Subclasse de SchemaData, define o esquema das bases sob o paradigma NoSQL Baseada em Grafos. De acordo com Nasholm (2012), bases de dados baseada em grafos é estruturada em grafos matemáticos. Um grafo G=(V, E) geralmente consiste em um conjunto de vertices V e um conjunto de arestas E. Uma aresta e ∈ E é um parte de vértices (v1, v2) ∈ V x V. Se o grafo é direto esses pares são ordenados. Os vértices do grafo são chamados de nós, e as arestas de relações. Cada nó contém um conjunto de propriedades. O trecho do código está ilustrado na Figura 4.5.

Figura 4.5 Classe Schema Data Graph

```
from NoSQL_Invetory.SchemaData import SchemaData


class SchemaDataGraph(SchemaData):

    def createSchema(self):
        node = raw_input("Insert node value: ")
        edge = raw_input("Insert edge value: ")
        vertice = raw_input("Insert vertice value:")
        self.putColumn([node, edge, vertice])
```

4.2.1.5 Esquema de Dados Chave Valor - SchemaDataKeyValue

Subclasse derivada do SchemaData onde define o esquema das bases sob o paradigma NoSQL Chave-Valor. Para Nasholm (2012), o modelo de dados Chave Valor é baseado na abstração de dados do tipo Map. Ele contém a coleção de pares chave-valor onde todas as chaves são únicas. O trecho do código está ilustrado na Figura 4.6.

Figura 4.6 Classe Schema Data Key Value

```
from NoSQL_Invetory.SchemaData import SchemaData


class SchemaDataKeyValue(SchemaData):

    def createSchema(self):
        key = raw_input("Insert key value: ")
        self.putColumn(key)
```

4.2.1.6 Sintaxe DDL - SyntaxDDL

Classe que utiliza o padrão factory method e define o comportamento da linguagem de definição de dados para cada tipo de SGBD e esquema de dados a serem criados, alterados ou excluídos. O trecho do código está ilustrado na Figura 4.7.

4.2.1.7 Sintaxe DDL Cassandra - SyntaxDDLCassandra

Subclasse de Sintaxe DDL onde define a linguagem de definição de dados do SGBD Cassandra para criação, alteração e exclusão de esquemas com dados oriundos do SGBD Cassan-

Figura 4.7 Classe Syntax DDL

```
from abc import abstractmethod

class SyntaxDDL:


    @abstractmethod
    def createSyntaxDDL(self):
        pass

    @abstractmethod
    def alterSyntaxDDL(self):
        pass

    @abstractmethod
    def dropSyntaxDDL(self):
        pass
```

dra.

4.2.1.8 Sintaxe DML - SyntaxDML

Classe que utiliza o padrão factory method e define o comportamento da linguagem de manipulação de dados para cada tipo de SGBD e esquema de dados a serem manipulados. O trecho do código está ilustrado na Figura 4.8.

Figura 4.8 Classe Syntax DML

```
from abc import abstractmethod

class SyntaxDML:

    @abstractmethod
    def selectSyntaxDML(self):
        pass

    @abstractmethod
    def updateSyntaxDML(self):
        pass

    @abstractmethod
    def insertSyntaxDML(self):
        pass

    @abstractmethod
    def deleteSyntaxDML(self):
        pass
```

4.2.1.9 Sintaxe DML Cassandra - SyntaxDMLCassandra

Subclasse de Sintaxe DML onde define a linguagem de manipulação de dados do SGBD Cassandra para a manipulação dos dados oriundos do SGBD Cassandra. Por meio dessa classe é possível buscar os dados da base de origem Cassandra.

4.2.1.10 Inventário - Inventory

Classe que define um inventário. Um inventário possui nome, descrição, dados de conexão e um conjunto de esquemas. O trecho do código está ilustrado na Figura 4.9.

Figura 4.9 Classe Syntax DML

```
class Inventory:

    def __init__(self, nameInventory, describeInventory, connection):
        self.schemas = []
        self.nameInventory = nameInventory
        self.describeInventory = describeInventory
        self.connection = connection

    def getInventoryName(self):
        return self.nameInventory

    def getInventoryDescribe(self):
        return self.describeInventory

    def getSchemas(self):
        return self.schemas

    def addSchema(self, schema):
        self.schemas.append(schema)
```

4.2.1.11 Fábrica de Inventário - InventoryFactory

Classe que cria inventários, ela é responsável por manter e remover inventários. O trecho do código está ilustrado na Figura 4.10.

Figura 4.10 Classe Syntax DML

```
from NoSQL_Invetory.Inventory import Inventory

class InventoryFactory:

    def __init__(self):
        self.inventories = []

    def addInventory(self, schemaData, nameInventory, describeInventory):
        self.inventories.append(Inventory(schemaData, nameInventory, describeInventory))

    def removeInventory(self, inventory):
        self.inventories.remove(inventory.getInventoryName())

    def getInventories(self):
        return self.inventories
```

4.2.1.12 Importação - Import

Classe responsável pela importação dos dados das bases NoSQL. Ela utiliza as classes de sintaxes, esquemas, acessa ao inventário e mantém os dados importados.

4.2.1.13 Amostra - Sample

Classe responsável por criar o esquema da amostra que será importada pela classe de importação. Ela acessa o inventário para utilizar o esquema de dados, sintaxes e dados de conexão para criar a amostra.

CAPÍTULO 5

Estudo de Caso

Para a aplicação da representação da modelagem do processo de ETL observar Trujillo and Mora (2003) e Lujan-Mora and Trujillo (2003) referenciado na dissertação de Mario.

Utilizar dois exemplos de bancos de dados HBase e Cassandra, ambos são de família de colunas, utilizar a mesma metodologia, testar, comparar e observar os resultados.

5.1 Descrição

5.2 Considerações Finais

CAPÍTULO 6

Conclusão

6.1 Principais Contribuições

6.2 Discussão

6.3 Resultados

6.4 Trabalhos Futuros

Referências Bibliográficas

[dS12] Mário Sergio da Silva. Um framework para desenvolvimento de sistemas etl. Master's thesis, Universidade Federal de Pernambuco, September 2012.

[Hei01] George T. Heineman. *Component-Based Software Engineering: putting pieces together*. Addison-Wesley, 2001.

[KR02] Ralph Kimball and Margy Ross. *The Data Warehouse Toolkit*. Robert Ipsen, 2002. Second Edition.

[LTP13] Xiufeng Liu, Christian Thomsen, and Torben Bach Pedersen. Cloudetl: Scalable dimensional etl for hive. *DB Tech Reports*, July 2013.

[Nas12] Petter Nasholm. *Extracting Data From NoSQL Databases*. PhD thesis, Chalmers University of Technology, SE-412 96 Goteborg Sweeden, January 2012.

[Sam97] Johannes Sametinger. *Software Engineering with Reusable Componets*. Springer, 1997.