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

Desde a década de 1970, com a criação do modelo relacional por Edgar Frank Codd, a estrutura de armazenamento adotada por muitos desenvolvedores de sistemas da área de tecnologia da informação tem se baseado no conceito de entidade e relação proposto por Codd. A maioria dos sistemas gerenciadores de banco de dados que possui aceitação no mercado fazem uso desse modelo, por exemplo o MySQL, Oracle e Microsoft SQL Server. Porém, os requisitos para o desenvolvimento de ferramentas de software modernas têm mudado significativamente, especialmente com o aumento das aplicações Web [Nas12]. 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 [Nas12]. O grande propósito das abordagens NoSQL é oferecer alternativas onde os esquemas relacionais não apresentam um bom desempenho. Esse termo abrange diferentes tipos de sistemas. Em geral, banco de dados NoSQL usam modelo de dados não-relacionais, com poucas definições de esquema, são executados em clusters e aplicados a alguns bancos de dados recentes como o Cassandra, o Mongo, o Neo4J e o Riak [FS13].

Muitas empresas coletam e armazenam milhares de gigabytes de dados por dia, no qual a análise desses dados torna-se uma vantagem competitiva no mercado. Por isso, há uma grande necessidade de uma nova arquitetura para o gerenciamento de suporte à decisão que possa alcançar melhor escalabilidade e eficiência [LTP13]. Para auxiliar no processo de gerenciamento de suporte à decisão uma das formas mais utilizadas é a criação de um ambiente data warehousing que é responsável por providenciar informações estratégicas e esquematizadas a respeito do negócio [CD97].

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.

Ferramentas de ETL são sistemas de software responsáveis por extrair dados de diversas fontes, transformar e customizar os dados e inseri-los no data warehouse. Comumente, esses processos são executados periodicamente, onde a otimização do seu tempo de execução torna-se importante [PAS05].

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, por algum tempo pouca importância foi dada ao processo de ETL 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].

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

[KR02]. Por isso, é comum que as ferramentas de ETL utilizadas no mercado atualmente só dêem suporte aos esquemas relacionais. Para oferecer suporte aos sistemas que necessitem utilizar um esquema não relacional de BDs NoSQL 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

Uma alternativa para organizar e manipular grandes volumes de dados sem utilizar um modelo relacional e ainda processá-los e armazená-los de maneira distribuída é fazer o uso de BDs NoSQL [dCS16]. Com isso, surge a necessidade de se promover meios para o uso desses BDs em DWs.

As pesquisas presentes na literatura 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.

O aumento do uso de banco de dados com esquemas não relacionais baseados no paradigma NoSQL e a falta de uma ferramenta programável, flexível e integrada, independente de plataforma que dê suporte à extração, transformação e carga em data warehouses para esses esquemas é a grande motivação deste trabalho.

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

1.3 Objetivos

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

1.3.1 Objetivo Geral

Esta pesquisa propõe o ETL4NoSQL que é um framework para desenvolvimento de aplicações ETL para sistemas que utilizam bases de dados NoSQL, com poucas definições de esquemas e não relacionais. Bases de dados NoSQL são utilizadas em muitas aplicações modernas, principalmente aplicações Web, onde possuem um grande volume e variedade de dados.

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 são 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, sugerimos a criação de um ambiente programável que oferecesse interfaces previamente selecionadas e permitisse a inserção de novas interfaces por meio de linguagem de programação. Essas interfaces são fundamentadas nos princípios de flexibilidade, reuso e inversão de controle adotadas pela literatura para desenvolvimento de frameworks [EC97] [V.03] [ECE99] [Sam97] [dS12]. Assim, a arquitetura do ETL4NoSQL dispõe de uma interface de programação com elementos para a importação dos dados das várias bases NoSQL, o mapeamento e manipulação desses dados para um modelo relacional que possa ser lido por qualquer ferramenta de ETL, possibilita também a criação de modelos não relacionais para o uso de DWs não relacionais, por meio da interface de programação, além de elementos que encapsulam os operadores de transformação, carga de dados que são fundamentos de ETL presentes na literatura [KC04] [dS12] [Hei01]. Para proporcionar a especialização dos elementos de importação e mapeamento do framework para diversos modelos de dados utilizamos uma abordagem

baseada em padrões, que possibilita a implementação do framework utilizando uma linguagem de programação de propósito geral.

Com isso, o objetivo principal desta pesquisa é especificar um framework programável, flexível e integrado para modelagem e execução de processos ETL em BDs NoSQL.

1.3.2 Objetivo Específico

O primeiro objetivo específico desta pesquisa é estender a proposta do framework para facilitar a carga de dados de dois sistemas de BD NoSQL distintos baseado no mesmo paradigma NoSQL em um DW relacional modelados pelo esquema estrela, tendo em vista que este é o esquema de dados dimensionais mais recomendado pela literatura [KR02] [**?**]. O outro objetivo específico é ao invés de dar carga em um DW relacional fazer uso do mesmo sistema em um DW NoSQL, seguindo a metodologia adotada por [CMK+15] em seu trabalho de pesquisa. Para isso, desenvolvemos dois frameworks especializados a partir do ETL4NoSQL em conformidade às peculiaridades dos processos de ETL nessas duas áreas de aplicação, os quais também são objetos de validação do ETL4NoSQL.

1.4 Justificativa

A integração de dados e os processos de ETL são procedimentos cruciais para a criação de data warehouses e sistemas BI (business intelligence). Porém, os sistemas para ETL e integração de dados são tradicionalmente desenvolvidos para dados estruturados em modelos relacionais que representam apenas uma pequena parte dos dados mantidos por muitas empresas [DBcRA05] [Russom 2007, Pedersen 2009]. Dessa forma, existe uma demanda crescente para integrar os dados não estruturados e semi estruturados em um repositório unificado. Devido a complexidade desses dados, novos desafios estão surgindo quando lidamos com dados heterogêneos e distribuídos no ambiente de integração [Salem, 2012].

Além disso, muitas empresas encontram dificuldades para lidar com as ferramentas ETL disponíveis no mercado. Aprender a lidar com essas ferramentas pode ser muito custoso em termos financeiros e de tempo, e por isso, acabam optando desenvolver os seus processos por meio de uma linguagem de programação de propósito geral [Awad et al., 2011; Muñoz et al., 2009].

Portanto, este trabalho propõe um framework programável para desenvolvimento de sistemas de ETL que possibilita a integração de dados não estruturados e semi estruturados armazenados em bases NoSQL. O framework possui um ambiente integrado para a importação e mapeamento dos dados, além da modelagem e customização dos processos de ETL. Os processos de importação e mapeamento do framework integram dados não estruturados e semi estruturados. Esses processos possibilitam a leitura e manipulação de dados de bases NoSQL, e também o armazenamento desses dados em bases deste tipo, oferecendo uma alternativa não relacional para a construção de DWs.

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, que consiste numa plataforma de software para desenvolvimento de sistemas ETL. Os dados de entrada de ETL4NoSQL podem ser de bases de dados não relacionais, e também de bases relacionais, mais especificamente, o framework permitirá a integração das diversas bases de dados NoSQL que pertencem aos quatro paradigmas de NoSQL: Orientada a documentos, Família de Colunas, Chave-Valor e Baseada em Grafos.

O ETL4NoSQL 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 proposto foram definidas as estruturas dos dados dos ambientes de origem, destino e da 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.

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

3.1 Requisitos do ETL4NoSQL

Requisitos de software são descrições de como o sistema deve se comportar, definidos durante as fases iniciais do desenvolvimento do sistema como uma especificação do que deveria ser implementado (SOMMERVILLE, 1997). Os requisitos podem ser divididos em funcionais e não funcionais, onde o primeiro descrevem o que o sistema deve fazer, ou seja, as transformações a serem realizadas nas entradas de um sistema, a fim de que se produzam saídas, já o outro expressa as características que este software vai apresentar.(SOMMERVILLE e SAWYER, 1997).

O ETL4NoSQL é um framework que tem como principal objetivo auxiliar na criação de processos de ETL ao se utilizar diversos tipos de bases de dados. Um sistema de software pode ter seus dados armazenados em bases relacionais, que seguem o modelo entidade e relacionamento, ou não relacionais, onde esta possui pouca definição de esquema, não segue um modelo específico e são chamados de NoSQL. As bases NoSQL possuem quatro paradigmas mais utilizados atualmente: Chave-Valor, Família de Colunas, Documentos e Grafos (REF).

As bases de dados relacionais utilizam uma linguagem de manipulação padrão conhecida por SQL (Structure Query Language), porém as bases de dados NoSQL não possuem uma linguagem em comum, como as relacionais, onde cada SGBD possui sua própria linguagem de manipulação de dados. Por isso, é essencial que haja um mecanismo que integre a leitura e escrita das bases NoSQL.

Processos de ETL possuem quatro etapas básicas: extração, limpeza/transformação e carga (Kimball and Caserta, 2004). O fluxo do processo de ETL inicia-se com a extração dos dados a partir de uma fonte, que podem ser bases de dados relacionais, bases NoSQL ou arquivos textuais. A partir da extração, os dados passam para uma Área de Processamento de Dados (APD), onde é possível executar processos de limpeza e transformação por meio de mecanismos de junção, filtro, união, agregação e outros. Finalmente, os dados podem ser carregados em estrutura de dados como repositórios analíticos, data warehouses, ou até mesmo em arquivos Linguagem de Marcação Flexível (XML).

Dessa forma, o ETL4NoSQL deve possuir um ambiente que importe os dados das diversas bases NoSQL, além das bases relacionais, e que possa fazer a leitura e escrita dos dados permitindo a execução dos processos de ETL. No quadro x.x apresentamos os principais requisitos elencados do ETL4NoSQL.

3.2 Arquitetura do ETL4NoSQL

Sommerville (2007), define o projeto de arquitetura como um processo criativo em que se tenta organizar o sistema de acordo com os requisitos funcionais e não funcionais. Um estilo de arquitetura é um padrão de organização de sistema (Garlan e Shaw, 1993; Sommerville, 2007), como uma organização cliente-servidor ou uma arquitetura em camadas. Porém, a arquitetura não necessariamente utilizará apenas um estilo, a maioria dos sistemas de médio e grande porte utilizam vários estilos. Para Garlan e Shaw, há três questões a serem definidas na escolha do projeto de arquitetura, a primeira é a escolha da estrutura, cliente-servidor ou em camadas, que permita atender melhor aos requisitos. A segunda questão é a respeito da decomposição

Tabela 3.1 Requisitos do ETL4NoSQL

Requisito	Prioridade
O sistema deve importar os dados de diversas bases relacionais e não relacionais	Alta
O sistema deve permitir a leitura e escrita dos dados importados	Alta
O sistema deve permitir mapear os dados no modelo relacional	Alta
O sistema deve permitir mapear os dados em quaisquer modelo desejado pelo usuário	Alta
O sistema deve possuir os mecanismos ETL mais conhecidos na literatura	Alta
O sistema deve possibilitar a criação de novos mecanismos ETL desejado pelo usuário	Alta
O sistema deve possuir um ambiente que possibilite a execução dos mecanismos de ETL em operações	Alta
O sistema deve permitir o reutilização dos seus mecanismos para vários cenários	Alta

dos subsistemas em módulos ou em componentes. E por fim, deve-se tomar a decisão de sobre como a execução dos subsistemas é controlada. A descrição da arquitetura pode ser representada graficamente utilizando modelos informais e notações como a UML (Clements, et al., 2002; Sommerville, 2007). A arquitetura do ETL4NoSQL, representada graficamente na figura x.x, é baseada no requisito de reutilização, a possibilidade do reuso, reduz o trabalho repetitivo na implementação de componentes e o custo de manutenção (Szyperski, et al.2002), e sua estrutura é em camadas, onde há a camada de sistema e a camada de interface. A camada de sistema lida com todas as operações internas e a camada de interface faz toda a interligação do sistema com o ambiente externo. A decomposição dos subsistemas do ETL4NoSQL é em componentes, pois componentes podem ser subsistemas ou simples objetos que podem ser reusados (Sommerville, 2007).

3.3 Componentes do ETL4NoSQL

A engenharia de software baseada em componentes é uma abordagem baseada em reuso para desenvolvimento de sistemas de software, ela envolve o processo de definição, implementação e integração ou composição de componentes independentes não firmemente acoplados ao sistema. Os componentes são independentes, ou seja, não interferem na operação uns dos outros e se comunicam por meio de interfaces bem definidas, os detalhes de implementação são ocultados, de forma que as alterações de implementação não afetam o restante do sistema (Sommerville, 2007). Segundo [Sam97], componentes são uma parte do sistema de software que podem ser identificados e reutilizados, onde descrevem ou executam funções específicas e

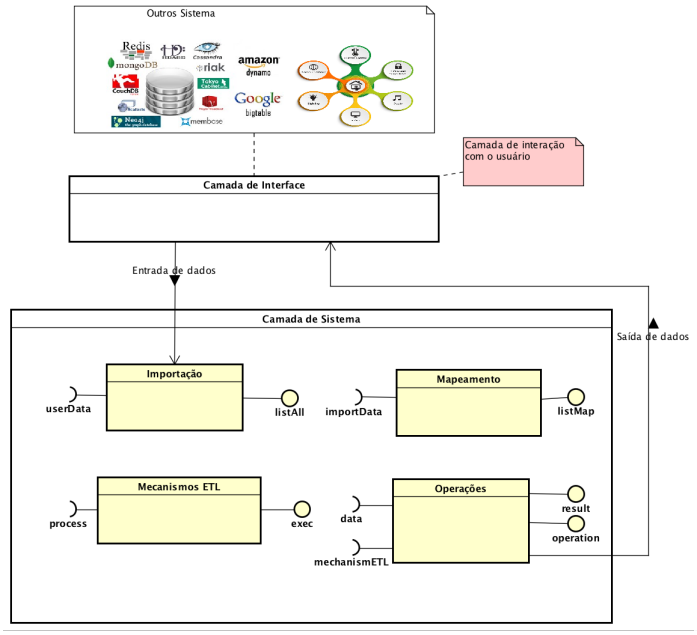


Figura 3.1 Arquitetura do Framework ETL4NoSQL

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. De acordo com os requisitos do ETL4NoSQL, foi possível identificar quatro importantes funcionalidades que podem ser definidas como componentes do sistema, a funcionalidade de importação, mapeamento de dados, mecanismos dos processos de ETL e o controlador de operações. Os componentes do ETL4NoSQL e suas características são apresentados nas seções a seguintes, seguindo as características de componentes adotadas por [Hei01].

3.3.1 Componente de Importação

Um dos objetivos do framework ETL4NoSQL é possibilitar a integração de vários tipos de bases de dados, relacionais ou não relacionais, presentes nos sistemas modernos. Para isso, a ferramenta deve permitir a leitura e escrita dos diversos SGBDs existentes. A solução encontrada para isso foi desenvolver um componente programável que possibilite a importação dos dados das bases por meio de inserção de parâmetros em linha de comando. Este componente, por ser criado utilizando o paradigma de orientação a objetos, permite também sua extensão, por meio de especialização, para que atenda a especificidade de cada cenário. As características do componente são apresentadas a seguir.

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

b) Nomeação: Import.

c) Metadados: Este componente contém as informações da base de origem como a linguagem de manipulação de dados e meios para estabelecer a conexão com a base, requer uma interação com a interface para o usuário disponibilizar as informações e fornece os dados importados para outros componentes.

d) Interoperabilidade: Oferece comunicação com outros componentes por meio dos métodos listAll e userData.

e) Customização: Este componente permite customizar as formas de apresentar os dados importados, de acordo com a necessidade de cada sistema.

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

g) Empacotamento e utilização: Os métodos são encapsulados e podem ser utilizados pela importação de sua classe e a interface com o usuário é por meio de linha de comando.

3.3.2 Componente de Mapeamento

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

b) Nomeação: Map.

c) Metadados: Este componente requer os dados de uma base de dados para efetuar o mapeamento.

d) Interoperabilidade: Oferece comunicação com outros componentes por meio dos métodos importData e listMap.

f) Customização: É possível customizar as regras de mapeamento para outros esquemas de dados.

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

h) Empacotamento e utilização:

3.3.2.1 Componente de Mecanismos de ETL

a) Interface: Componente que contém métodos que realizam as principais operações de ETL presentes na literatura.

b) Nomeação: MechanismETL.

c) Metadados: Este componente requer dados de controle para realizar as operações por meio de seus métodos.

d) Interoperabilidade: Oferece comunicação com outros componentes por meio dos métodos exec e process.

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

3.3.2.2 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

3.4 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/

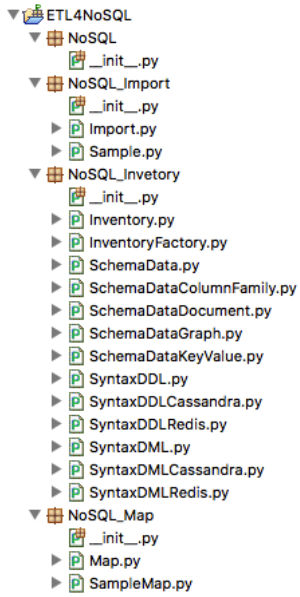
4.2 Interfaces de Programação

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 $\in$ E é um parte de vértices (v1, v2) $\in$ 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

Este capítulo provê aplicações exemplos baseadas no ETL4NoSQL em dois domínios distintos para um mesmo problema que é o controle de logs de um servidor de controle de registros acadêmicos. O desenvolvimento dessas aplicações almeja ilustrar a reusabilidade e flexibilidade do framework para vários tipos de bases não relacionais.

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

[CD97] S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. *SIG- MODRecord,v.26,n.1,p.65â74*, 1997.

[CMK+15] Max Chevalier, SMohammed El Malki, Arlind Kopliku, Olivier Teste, and Ronan Tournier. Implementing multidimensional data warehouses into nosql. *Proceedings of the 17th International Conference on Enterprise Information Systems, p.172-183, April 27-30*, 2015.

[DBcRA05] Jerome Darmont, Omar Boussaid, AJean christian Ralaivao, and Kamel Aouiche. An architecture framework for complex data warehouses. *7th International Conference on Enterprise Information Systems (ICEISâ05), Miami, USA, pages 370â373*, 2005.

[dCS16] Lucas de Carvalho Scabora. AvaliaÃ§Ã£o do star schema benchmark aplicado a bancos de dados nosql distribuÃdos e orientados a colunas. Master's thesis, USP - SÃ£o Carlos, April 2016.

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

[EC97] FAYAD M. E. and SCHMIDT D. C. Object-oriented application frameworks. *Communica- tions of the ACM 40 (10): 32â38*, 1997.

[ECE99] FAYAD M. E., SCHMIDT D. C., and JOHNSON R. E. Building application frameworks: object-oriented foundations of framework design. *John Wiley and Sons, New York, USA, pp. 3â29*, 1999.

[FS13] Martin Fowler and Pramod J. Sadalage. *NoSQL Essencial Um guia conciso para o mundo emergente de PersistÃªncia Poliglota*. Novatec, 2013. Primeira EdiÃ§Ã£o.

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

[KC04] Ralph Kimball and J. Caserta. *The Data Warehouse ETL ToolKit*. Robert Ipsen, 2004.

[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.

[PAS05] VASSILIADIS P., SIMITSIS A., and GEORGANTAS P. TERROVITISB M. SKIADOPOU-LOS S. A generic and customizable frameworkfor the design of etl scenarios. *Information Systems - Special issue: The 15th international conference on advanced information systems engineering 30 (7): 492â525*, 2005.

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

[V.03] BRAGA R. T. V. Um processo para construÃ§Ã£o e instanciaÃ§Ã£o de frameworks baseados em uma linguagem de padrÃµ*esparaumdomnioespecfico*.*Master*′*sthesis*,*ICMC*/*USP*,*S£oPaulo*,2003.