UNIVERSIDADE FEDERAL DO PARANÁ

FELIPE SHI IU WU

ADAPTADOR DE BANCO DE DADOS TEXTUAL PARA POLYSTORE ANALÍTICO

CURITIBA

2019

FELIPE SHI IU WU

ADAPTADOR DE BANCO DE DADOS TEXTUAL PARA POLYSTORE ANALÍTICO

Trabalho apresentado como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação no curso de Ciência da Computação, Setor de Ciências Exatas da Universidade Federal do Paraná.
Orientador: Prof. Dr. Marcos Didonet Del Fabro

CURITIBA

2019

TERMO DE APROVAÇÃO

FELIPE SHI IU WU

ADAPTADOR DE BANCO DE DADOS TEXTUAL PARA POLYSTORE ANALÍTICO

Trabalho apresentado como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação no curso de Ciência da Computação, Setor de Ciências Exatas da Universidade Federal do Paraná., pela seguinte banca examinadora:

Prof. Dr. Marcos Didonet Del Fabro
Orientador

Professor
UFPR

Professor

Professor

Curitiba, Julho de 2019.

Este trabalho é dedicado
aos bombeiros que, quando pequenos, sonharam em se tornar bombeiros;
aos astronautas que, quando pequenos, sonharam em se tornar astronautas e
aos cientistas que, quando pequenos, sonharam em se tornar cientistas.

AGRADECIMENTOS

Os agradecimentos iniciais são dedicados aos meus colegas da graduação que foram essenciais para que chegue nesse estágio do curso. Em seguida, meus agradecimentos especiais para:

- aos meus pais, Wu Cheng Po e Chang Chiu Lan, sem a confiança deles eu não teria tanta liberdade durante a minha jornada na graduação;
- ao meu orientador, Prof. Dr. Marcos Didonet Del Fabro, pela competência, incentivo e companheirismo durante o desenvolvimento do presente trabalho;
- aos meus colegas e amigos Lucas Fernandes de Oliveira e Alessandro Elias pela paciência, pois além de participantes do projeto Blendb, me ajudaram em várias dificuldades encontradas no caminho;
- ao meu colega e amigo Lucas Olini, com quem não só enfrentei a jornada de quatro anos e meio até aqui, como esteve ao meu lado em quase todas as fases de decisão nesse curso;
- à minha colega e namorada Gabriela Yukari Kimura, que me acompanhou em tantas manhãs difíceis que me dediquei à esse trabalho.

"Não como um caos, ferido e esmagado junto, mas, como o mundo harmoniosamente confuso:
Onde a ordem nós vemos em diversidade. (Alexander Popo, Windsor Forest)

RESUMO

Gradualmente o número de modelos e sistemas de gerenciamento de banco de dados vêm aumentando. Chegando ao ponto no qual diversos bancos de dados são adotados em um único sistema. Vários bancos com várias linguagens de consultas diferentes aumentam a complexidade do sistema como um todo. Além da grande quantidade de dados abertos disponíveis, os quais requerem métodos otimizados para manipular de modo eficiente. A proposta abordada nesse trabalho é relacionado à sumarizar as consultas, de modo que, mais bancos integrados não signifique mais complexidade para realizar buscas.

A partir da linguagem simplificada disponibiliza pelo Blendb, apenas com as métricas, dimensões e filtros podemos realizar todas as consultas analíticas. Sobre a arquitetura atual do Blendb, no qual comporta apenas banco de dados relacionais. Implementar um componente adaptador com a funcionalidade de construir consultas Elasticsearch através de uma linguagem de fácil entendimento ao usuário e única para todos os bancos integrados.

O adaptador foi implementado e testado, abrangendo a maior parte das funcionalidades disponibilizadas pela API. Além da implementação, o presente trabalho demonstra grande parte dos cuidados necessários para que os obstáculos da construção do adaptador sejam facilmente superados. Espera-se que melhorias no Blendb em geral tragam mais funcionalidades e que com isso, seja necessário que as novas funções sejam implementadas para que o Elasticsearch faça parte do leque de banco de dados suportados pela API.

Palavras-chaves: Polystore. Modelo orientado a documento. Elasticsearch. Consultas em banco de dados. Consultas analíticas

ABSTRACT

The number of database management models and systems has been increasing gradually. To the point where several databases are adopted in a single system. Many databases with several different query languages increase the complexity of the system as a whole. In addition to the large amount of available open data, which require optimized methods to process efficiently. The proposal present in this work is related to summarizing the queries, so that more integrated databases do not mean more complexity to search.

From the simplified language provided by Blendb, with only metrics, dimensions and filters we can perform all analytical queries. On the current architecture of Blendb, which contains only relational database. Implement an adapter that builds Elasticsearch queries through an easy-to-understand user language and unique to all integrate databases.

The adapter has been deployed and tested, covering most of the functionality provided by the API. In addition to the implementation, the present work demonstrates much of the necessary care so that the obstacles of the construction of the adapter are easily overcome. It is expected that improvements in Blendb will bring more functionality and with this, it will be necessary the implementation for the new functions for Elasticsearch to be part of the databases supported by the API.

Key-words: Polystore. Document-oriented database. Elasticsearch. Database queries. Analytics queries

LISTA DE ILUSTRAÇÕES

LISTA DE CÓDIGOS

LISTA DE ABREVIATURAS E SIGLAS

ANSI ..... Instituto Americano de Padronização
C3SL ..... Centro de Computação Científica e Software Livre
Inep ..... Instituto Nacional de Estudos e Pesquisas
ISTC ..... Intel Science and Technology Center
OLAP ..... Processamento Analítico Online
OLTP ..... Processamento de Transação Online
SGBD ..... Sistema de Gerenciamento de Bancos de Dados
SQL ..... Linguagem de Consulta Estruturada
UFPR ..... Universidade Federal do Paraná

SUMÁRIO

1 INTRODUÇÃO

A crescente variedade de sistemas de bancos de dados é um sinal de como eles estão se personalizando para cada ambiente em que atua, consequentemente, cada ambiente adota a melhor opção para sua aplicação (STONEBRAKER, 2005).

Suponha que um ambiente de armazenamento de dados tenha sido construído de maneira eficiente para atender aos seus diversos sistemas, e que para isso, foram necessários adotar diversos bancos de dados adequados a cada serviço. No momento que a organização responsável desejar realizar consultas sobre esse ambiente com armazenamento distribuído em diversos bancos e sistemas de gerenciamento. A dificuldade para coordenar as consultas e requisições de forma manual seria muito grande e não escalável, além de ser inviável a longo prazo.

Visando atender a necessidade de gerenciar os dados e obter informações armazenadas em diversos bancos, o cientista da computação caminha na direção de pesquisar e construir novos mecanismos para prover uma melhor interação entre o usuário e a tecnologia, como por exemplo, oferecendo uma interface única e menos complexa (STONEBRAKER, 2005).

Stonebraker (2005) já dizia que o paradigma "um tamanho serve para todos" não seria mais aplicável para a onda de novos mecanismos de dados especializados. Eles argumentaram que mecanismos especializados podem oferecer um desempenho mais econômico e, atualmente, a sua visão está mais válida do que nunca. Prova disso, são os diversos sistemas de dados especializados se tornando cada vez mais populares.

Os *polystores* são exemplos de projetos que integram vários sistemas de gerenciamento de banco de dados sob um só. Como por exemplo *Apache Calcite* em Begoli et al. (2018), *Apache Drill* em Hausenblas e Nadeau (2013) ou o *BigDAWG* em Duggan et al. (2015). Entretanto, nenhuma das propostas possuem o foco em consultas analíticas, como é o caso do projeto Blendb.

O Centro de Computação Científica e Software Livre (C3SL) da Universidade Federal do Paraná (UFPR) possui diversos projetos, e entre eles, está o Blendb. Uma ferramenta desenvolvida para facilitar as consultas analíticas nas diferentes bases de dados abertos coletadas das máquinas instaladas em escolas, telecentros e quilombos. O projeto Blendb ainda está em desenvolvimento, sendo assim, vários sistemas de gerenciamento de banco de dados ainda não foram integrados e dentre eles o *Elasticsearch*. Sendo assim, o objetivo do presente trabalho é integrar um novo sistema de gerenciamento de banco de dados ao Blendb, abrindo portas para um modelo não relacional, que é o caso do Elasticsearch.(C3SL, 2019).

A escolha de integrar o *Elasticsearch* ao Blendb se deve à sua expressiva presença na linha de pesquisa textual e na contribuição para encontrar dados importantes em um grande emaranhado de documentos. O C3SL já utiliza o *Elasticsearch* para armazenar alguns de seus

dado, deste modo, se torna profícuo a inclusão do *Elasticsearch* ao leque de sistemas de bancos de dados que comunicam com o Blendb.

O restante do artigo está organizado da seguinte forma: a Seção II aborda os conceitos de bancos de dados necessários para a análise da proposta e da sua implementação. A Seção III descreve propostas semelhantes ao projeto Blendb. A Seção IV descreve a proposta em detalhes e o processo de implementação. Na Seção V trazemos resultados obtidos ao aplicarmos a proposta implementada. Finalmente, concluímos e discutimos os espaços para trabalhos futuros na Seção VI.

2 REFERENCIAL TEÓRICO

Antes de se aprofundar na proposta do trabalho é necessário se situar dentro dos livros e das documentações. Em outras palavras, a partir de quais pretextos estamos realizando o presente trabalho. Um adaptador que traduz uma certa linguagem para a consulta nativa do sistema de banco de dados, por mais pequeno que seja esse componente na arquitetura do sistema, ela requer alguns conhecimentos teóricos. Nesta seção será apresentado as informações que contribuem para o entendimento do trabalho.

2.1 BANCO DE DADOS

Um *dado* se refere à informação após armazenada em um banco de dados, que por sua vez, é um repositório eletrônico que armazena esses dados computadorizados e reage às requisições dos usuário através da camada de software conhecida como Sistema de Gerenciamento de Bancos de Dados (SGBD)(DATE, 2003).

É importante diferenciar **sistemas de gerenciamento de banco de dados** do **banco de dados** propriamente dito. O SGBD é o software que atua como gerenciador do dado, controlando as operações do usuário sobre os dados, o qual, através dele que permite ao usuário armazenar informações no banco, buscar esses dados ou atualizá-las.

2.1.1 Banco de Dados Relacional

O banco de dados relacional é definido de um modo abreviado pelo Date (2003, p.23) como aquele no qual:

1. Os dados são percebidos pelo usuário como tabelas (e nada além de tabelas).
2. Os operadores à disposição do usuário (por exemplo, para busca de dados) são operadores que geram tabelas “novas” a partir de tabelas “antigas”. Por exemplo, há um operador *restrição* para extrair um subconjunto das linhas de uma dada tabela, e outro operador, *projeção*, que extrai um subconjunto das colunas – e, é claro, um subconjunto de linhas e um subconjunto de colunas de uma tabela podem ambos, por sua vez, ser considerados tabelas.

Vale dizer que o operador restrição citado pelo Date (2003) é também conhecido como operador seleção.

Se falamos sobre sistemas relacionais, estudamos também a Linguagem de Consulta Estruturada (SQL), que além de ser uma linguagem de pesquisa declarativa amplamente utili-

zada nos grandes SGBD relacionais, atualmente é um padrão definido pelo Instituto Americano de Padronização (ANSI) (KORTH, 1991).

2.1.2 Banco de Dados Orientado a Documento

No cenário atual, os dados estão se tornando cada vez mais complexos, deixando de ser simplesmente uma lista de chave e valor, passando a conter estruturas como datas, localizações geográficas, outros objetos ou até mesmo lista de valores. Usando um banco relacional significaria armazenar estruturas complexas de dados em linhas e colunas, ajustando a um esquema da tabela e reconstruí-la toda vez que for necessário utilizar o dado em sua estrutura original. (GORMLEY, 2015)

Uma das propostas que temos para armazenamento dos dados diferente do modelo relacional é o banco de dados orientado a documento. Destinado para ambientes com operações que exigem um esquema flexível para desenvolvimento rápido e iterativo. Além da flexibilidade, o modelo é utilizado para facilitar a extração de informações em *big datas*. Podemos citar como exemplo a Amazon DocumentDB (AMAZON, 2019).

Em geral, o documento consiste em dados encapsulados e codificados em alguns formatos ou codificações padrões, os mais famosos são XML, YAML, JSON e BSON.

Os documentos não são obrigados a aderir a um esquema padrão, e nem todos precisam conter as mesmas chaves. Armazenamentos de documentos são similares às estrutura chave-valor em programação, o qual permitem que tipos de documentos diferentes fiquem em um único armazenamento, além disso, permitem que os campos dentro deles sejam opcionais.

2.1.3 Processamento Analítico Online - OLAP

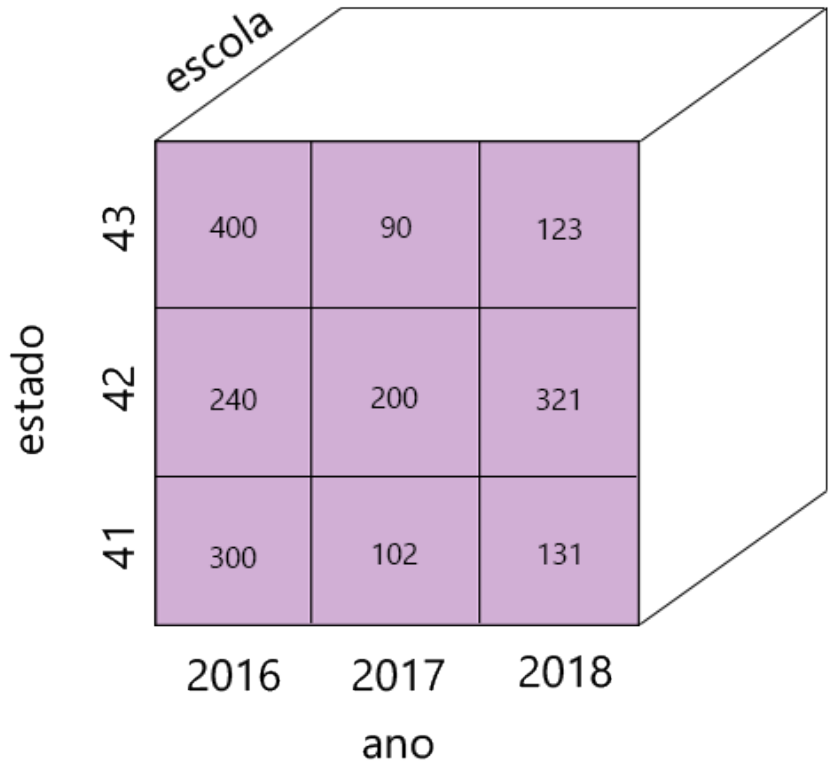
Até o momento foi descrito diferentes formas de armazenar os dados. Outra característica relevante para selecionar um é a carga que será executada sobre ele, isto é, quais as operações mais frequentes e suas características.

Aplicações que realizam alterações (por exemplo, depositar um cheque ou sacar um valor da conta) e um grande número de transações que exigem confiabilidade e eficiência, tais aplicações de Processamento de Transação Online (OLTP) ocupavam grande parte do foco dos SGBDs e já foram as principais aplicações dos modelos relacionais.

Entretanto, os bancos de dados ganharam cada vez mais espaços, em todos os sentidos, por conta da demanda. Surgiu o cenário em que se armazena dados com papéis não somente para documentar operações, mas também históricos das mais diversas maneiras. Tais dados são extensos e essa característica se torna essencial para as novas aplicações de apoio à decisão e que geram resumos de análise estatística e exploração de dados, utilizados para ajudar a "definir tendências, apontar problemas e tomar... decisões inteligentes" Date (2003, p.590).

É no domínio da análise de dados que encontramos o Processamento Analítico Online (OLAP), definido por Date (2003, p.607) como “o processamento interativo de criar, gerenciar, analisar e gerar relatórios sobre dados”. Para um melhor entendimento do OLAP, é necessário introduzirmos o conceito de modelo de dados multidimensional. Conhecido também como *array multidimensional*, as consultas sobre esse modelo respondem uma coleção de **medidas** numéricas para um conjunto de **dimensões**. Utilizando um exemplo para auxiliar na análise, podemos usar como atributo de medida *o número de matrículas* das escolas do Brasil e as dimensões são *escola, estado e ano*. Dado uma escola, um estado e um ano, teremos um valor numérico associado a esse conjunto. No exemplo da figura 1, para uma determinada escola, o número de matrículas do estado *41* no ano de *2018* é *131*. (RAMAKRISHNAN, 2007).

FIGURA 1 – Matrículas: uma representação multidimensional



Ainda sobre dimensões, podemos acrescentar que cada atributo da dimensão pode ainda ser associado a um conjunto de outros atributos. Por fim, é importante ressaltar que o apoio à decisão não faz parte da tecnologia de banco de dados, são usos específicos e que demandam estruturas que alguns SGBDs podem facilitar nos seus processos.

2.1.3.1 Índices invertidos

Do inglês *inverted index*, os índices invertidos são uma estrutura de dados que mapeia termos às suas ocorrências em um documento ou conjunto de documentos. É uma estratégia de indexação que permite a realização de pesquisas textuais de forma rápida (ELASTIC, 2019).

Um índice invertido consiste em uma lista de todas as palavras únicas que aparecem em qualquer documento e, para cada palavra, uma lista dos documentos em que aparece. Para entender de forma mais fácil, iremos usar um exemplo contendo dois *documentos*, o conteúdo delas são:

1. Até mesmo a menor das pessoas pode mudar o rumo do futuro
2. Até mesmo o mais sábio das pessoas aprendeu com os erros

Uma das implementações possíveis é dividir o conteúdo dos *documentos* em *termos*, que são cada uma das palavras dos documentos. Criamos uma lista de *termos* sem repetição, e ao lado de cada *termo*, colocamos os *documentos* que contém esses *termos*. A tabela 1 demonstra a indexação no nosso exemplo, já a tabela 2 mostra o resultado da busca por "*menor das pessoas*". O valor *Total* apresentado na tabela 2 é a quantidade de palavras encontradas em cada documento ao realizar a busca, o qual é utilizado para ordenar os documentos com maior acerto e desta maneira aumentar as chances de busca. (GORMLEY, 2015)

TABELA 1 – Demonstração do índice invertido

Termo	Documentos
Até	{1, 2}
mesmo	{1, 2}
o	{1, 2}
menor	{1}
das	{1, 2}
pessoas	{1, 2}
pode	{1}
mudar	{1}
rumo	{1}
do	{1}
futuro	{1}
a	{1}
mais	{2}
sábio	{2}
aprendeu	{2}
com	{2}
os	{2}
erros	{2}

TABELA 2 – Resultado da busca no índice invertido

Termo	Doc_1	Doc_2
menor	x	
das	x	x
pessoas	x	x
Total	3	2

2.2 ELASTISCSEARCH

Elasticsearch é um sistema de gerenciamento de banco de dados orientado a documentos escrito em Java e construído sobre o Apache Lucene. O SGBD vem ganhando fama devido à sua escalabilidade em termos de volume de dados e taxa de transferência de consulta (query throughput), uma vez que ela é amplamente distribuível. Diversas grandes empresas como Facebook, GitHub e o Firefox utilizam o Elasticsearch, por conta disso, o Elastic possui uma grande comunidade ativa (GORMLEY, 2015).

2.2.1 Elastic Stack

Elastic Stack é um conjunto de tecnologias, incluindo o Elasticsearch, os quais oferecem um pacote de ferramentas que auxiliam a utilização do Elasticsearch. As ferramentas do Elastic Stack utilizadas nesse trabalho são:

- *Kibana*: uma plataforma de análise e visualização, que permite visualizar facilmente os dados do Elasticsearch. É o painel de controle visual utilizado para visualizar os dados, inclusive em forma de gráficos, e testar as consultas. Além de ser possível adicionar dados no Elasticserach diretamente via Kibana.
- *Logstash*: processa registros de aplicações e envia para o Elasticsearch, mas se sofisticou tornando um processador de dados em pipeline. Utilizado para inserir dados em CSV.

E por fim o próprio Elasticsearch, um SGBD que armazena os dados em uma unidade básica de informação armazenável chamada de *Documento*. O *documento* é essencialmente um objeto JSON, e corresponde a uma linha em um banco de dados relacional. As *chaves* dentro do objeto são equivalentes às colunas do banco relacional. Já as consultas são realizadas via API REST, e também utiliza o formato JSON. O código 2.2.1 exemplifica uma consulta nativa do Elasticsearch que encontra o maior valor da chave *ano* agrupado por *estado*, ordenando de forma ascendente.

2.2.2 Arquitetura

A forma que o Elasticsearch é construído difere muito dos bancos de dados relacionais, portanto, alguns conceitos básicos são necessários para entender o seu comportamento. Neste trabalho é importante apenas entender as funcionalidades chaves, logo não iremos aprofundar demasiadamente.

O Elasticsearch é altamente distribuível, cada instância do Elasticsearch é chamado de *node*, e esta instância fica indexado dentro de um *cluster*. A funcionalidade principal do *node* é armazenar os dados e a do *cluster* é indexar este conjunto de *nodes* para prover as buscas dos dados dentro de cada um deles (GORMLEY, 2015).

```
{
  "aggs": {
    "group_by_estado": {
      "terms": {
        "field": "estado.keyword",
        "size": 1000,
        "order": {
          "max_ano": "asc"
        }
      },
      "aggs": {
        "max_ano": {
          "max": {
            "field": "ano"
          }
        }
      }
    }
  }
}
```

CÓDIGO 2.2.1: Exemplo de consulta Elasticsearch

Como já mencionado, os dados no Elasticsearch são armazenados como *documentos*, acrescentado a isso, existe o *índice*. De forma breve, o *índice* é uma coleção de *documentos* com características semelhantes, divididos e armazenados em *fragmentos*[1] dentro dos *nodes* (GORMLEY, 2015).

2.2.3 Elasticsearch: SQL Translate API

O Elasticsearch traz também uma API que realiza a tradução de comandos SQLs para consultas nativas do Elasticsearch. O código 2.2.2 apresenta a forma como a API pode ser utilizada, no entanto, é importante notar que *joins* não são aceitos. O uso da API se destina a realizações de consultas direcionadas ao Elasticsearch, sendo assim, é inviável realizar a mesma consulta SQL dos bancos relacionais. O Elasticsearch não possui o mesmo esquema de tabelas e não responderá como desejado (ELASTIC, 2019).

```
POST /_sql/translate
{ "query": "SELECT * FROM matricula ORDER BY estado DESC" }
```

CÓDIGO 2.2.2: Estrutura base da consulta Elasticsearch

[1] *Fragmentos* são estruturas do Elasticsearch utilizados para distribuir o armazenamento dos documentos entre as máquinas (GORMLEY, 2015)

2.3 BLENDB

Sobre o pretexto de "um tamanho não serve para todos" (STONEBRAKER, 2005). Começou a busca pelos desenvolvimento de soluções que interajam entre os SGBD. Com a existência de várias linguagens de consulta e diferentes arquiteturas dos SGBD, novas pesquisa propuseram o conceito de *polystore*.

Polystores possuem como característica possibilitar o funcionamento simultâneo de mecanismos de banco de dados e modelos de dados diferentes. Suportando as funcionalidade dos SGBDs integrados enquanto diminui a complexidade necessária para realizar as operações com os dados. Já existem várias propostas de polystores sendo desenvolvidas pelo mundo, com várias arquiteturas diferentes. No entanto, o objetivo comum é não aumentar a complexidade de comunicação do usuário com o polystore mesmo que aumente a quantidade de SGBDs integrados e possibilitar interação entre os SGBDs integrados de forma eficiente (GADEPALLY et al., 2016).

O Blendb é um projeto do C3SL que visa implementar um *polystore* para facilitar as consultas sobre os dados abertos coletados ou recebidos de todo o Brasil. Atualmente opera como uma interface para facilitar a realização de consultas analíticas para os SGBDs *PostgreSQL* e *MonetDB*. O Blendb possui uma documentação enxuta e com fácil instalação, sendo assim, esta sessão irá apenas esclarecer alguns pontos que facilitam a análise da proposta deste trabalho. Primeiramente sobre os arquivos de configuração:

2.3.1 Arquitetura

Iniciando pelos arquivos de configuração, uma vez que as consultas construídas pelo Blendb se baseiam completamente nos mesmos. O arquivo `config.env` faz a configuração das informações básicas que possibilitam a comunicação do Blendb com o SGBD desejado. Atualmente, não foi implementado a opção de consultar diversos SGBDs simultaneamente. É necessário configurar qual SGBD o Blendb deve fazer a busca e como deve ser feito essa comunicação, ambos configurados no arquivo *.env*. Já o arquivo `config.yaml` descreve o esquema do banco de dados, informando quais são as agregações existentes e quais métricas e dimensões cada agregação contém (C3SL, 2019).

2.3.1.1 API

A API é a porta de comunicação com o Blendb, sua função é receber os comandos do usuário, traduzir o formato de texto em um formato interno, chamado de *view* e acionar os meta-dados corretos e necessários para construir e realizar a consulta. Essa API RESTful[2], possui 3 rotas. Uma que descreve as métricas disponíveis, uma que descreve as dimensões

[2] RESTful: capacidade de determinado sistema aplicar o conjunto de princípios da arquitetura REST.

disponíveis e outra para realizar consultas. As respostas da API são em formato JSON, e as possibilidades são duas, ou o resultado da consulta ou uma mensagem de erro (C3SL, 2019).

2.3.1.2 Engine

A engine é o componente que descobre e seleciona as agregações que serão utilizadas na consulta. Esse componente utiliza a descrição encontrada no arquivo config.yaml .

Uma vez que a engine recebe uma consulta, ela usa o conhecimento do esquema do banco para selecionar as agregações. Ela também tem a capacidade de informar se uma consulta não pode ser realizada. A engine também informa para a API quais são as métricas e dimensões disponíveis.

2.3.1.3 Adapter

O adapter é o componente do Blendb que recebe a consulta na estrutura interna *view* contendo métricas, dimensões, filtros e o conjunto de agregações. O componente reconstrói a consulta no formato nativo do SGBD específico, sendo assim, cada banco de dados precisaria ter um adaptador próprio dentro do Blendb (C3SL, 2019).

O adapter é a peça fundamental para o Blendb se comunicar com o banco de dados. Pois além de traduzir a consulta recebida pela API, o adapter realiza a conexão com o SGBD e executa a consulta reconstruída, gerando um retorno padrão e devolvendo à API.

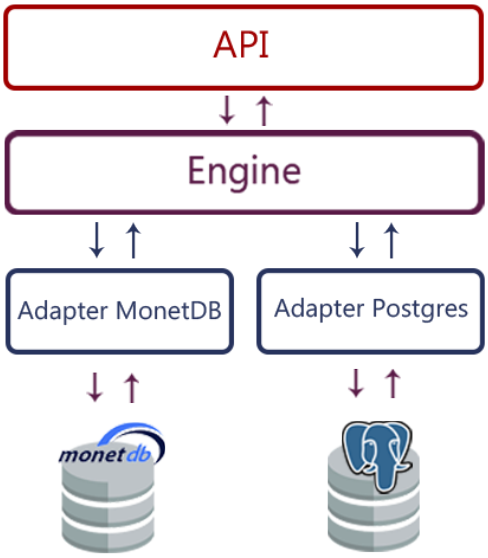
Esse é o único componente que interage diretamente com o banco de dados. Toda interação do Blendb com um banco de dados deve ser feita através de um adapter. Como os outros componentes, API e *engine* são independentes do SGBD, para incluir um novo sistema de banco de dados ao Blendb, apenas o adapter precisa ser implementado. Os outros componentes podem ser reaproveitados e apenas o adapter necessário precisa ser adicionado. Atualmente o Blendb conta com dois adapters, um que comunica com o MonetDB e outro com o PostgreSQL. A figura 2 descreve graficamente a arquitetura do Blendb.

2.3.2 Consulta

Para realizar consultas através do Blendb são necessários apenas 3 parâmetros. O seguinte formato é esperado: inicia com **data?**, seguido de **metrics=** e uma lista de métricas divididas por vírgulas.

A dimensão vem a seguir e segue o mesmo padrão que as métricas, só é necessário dividir com um **&**, desta maneira **&dimensions=**. Já os filtros, precisam de um operador de comparação, entretanto, o padrão segue o mesmo. Separado com um **&** e divide a lista de filtros com vírgulas para **OU** e ponto-vírgula para **E**, assim **&filters=field!=2**. Um exemplo de consulta completa ficaria:

FIGURA 2 – Arquitetura do Blendb




```
data?metrics=AVG_AGE&dimensions=UF,SEXO&filters=SEXO==2,UF!=41
```

2.3.2.1 Métricas

Uma métrica é a informação indireta do banco de dados gerado por uma função de agregação. As funções de agregação disponíveis do Blendb são: média, máximo, mínimo, soma e contagem. Logo, as métricas são sempre valores numéricos. Uma métrica está sempre associada a uma função específica. Um exemplo de métrica é a média de idade dos alunos (C3SL, 2019).

Durante uma consulta, o usuário pode estar interessado na média das idades por estado. Para realizar a consulta, os dados devem ser combinados em um único registro por estado. Para combinar esses registros, a função média seria utilizada. É importante destacar que no Blendb, uma métrica fica sempre associada a uma única função de agregação. Isto é, caso seja configurado que a idade dos alunos seja atribuído à função de média, não será possível realizar a soma das idades sem alterar o arquivo de configuração. De acordo com a documentação, a métrica é obrigatória ao se realizar uma consulta através do Blendb (C3SL, 2019).

2.3.2.2 Dimensões

Uma dimensão determina o grau de detalhe que se deseja visualizar uma métrica. Quando nenhuma dimensão é fornecida, o Blendb retornará no máximo um registro contendo a agregação de todos os registros do banco de dados. Caso nenhum registro tenha passado pelos filtros, o retorno será o valor zero.

A dimensão é o agrupamento dos dados. Como por exemplo na consulta “média das idades por estado”, as dimensões definem os grupos. No caso *estados* define a quantidade de

grupos. O resultado da consulta seria um valor numérico para cada estado, no entanto, no Blendb, caso 0 registros forem agrupados, o grupo não será retornado. A dimensão não é obrigatória na consulta mais precisa deixar **dimensions=**, mesmo que seja seguido de nenhum campo (C3SL, 2019).

2.3.2.3 Filtros

Um filtro é uma restrição sobre a consulta. Ele restringe quais dados serão enviados para serem agregados, sendo assim, são aplicados sobre as dimensões. Na consulta "média das idades de alunos do sexo tipo 2, por estado", ao invés de se considerar todos os registros, apenas os alunos do sexo tipo 2 serão considerados e enviados para serem agregadas.

Os filtros possuem as seguintes partes: dimensão, operação, valor. As operações incluem maior que, menor que, maior ou igual, menor ou igual, igualdade e diferença. As operações são limitadas pelo tipo de dados, por exemplo, não realiza comparações de *maior* sobre valor booliano (C3SL, 2019).

3 TRABALHOS RELACIONADOS

Para solucionar as consequências acarretadas por esta diversificação de mecanismos de dados, algumas frentes de pesquisas trouxeram propostas interessantes e que abriram um leque de possibilidades para os novos horizontes.

A comunidade de pesquisa também avança na área de processamento de consultas distribuídas, transações distribuídas e gerenciamento de réplica. SGBDs distintos colocados em um mesmo ambiente se torna cada vez mais vantajoso a medida que comunicação entre os bancos se torna mais consistente e completa.

3.1 BIG-DAWG

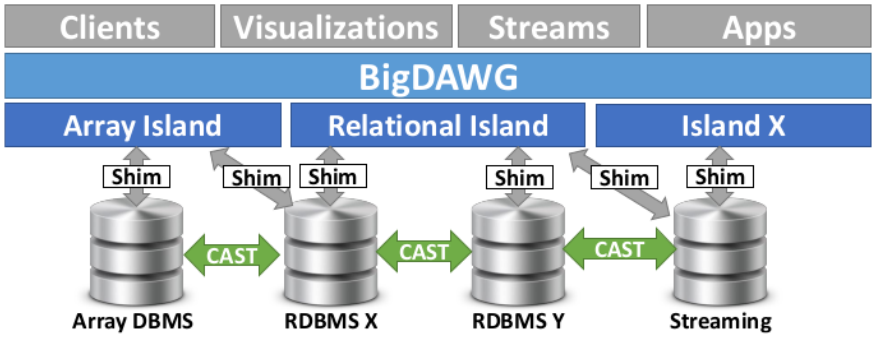
Um exemplo de pesquisa no campo de sistema de armazenamento e processamento de dados integrados temos o Big-DAWG. Um projeto de código aberto desenvolvido pela Intel Science and Technology Center (ISTC) para trabalhar com *Big Data*.

O Big-DAWG é uma polystore que Gadepally et al. (2016) define como: uma ferramenta de interface unificada para os usuários, construída sobre múltiplos, heterogêneos e integrados sistemas de gerenciamento de dados subjacentes de forma transparente com o intuito de simplificar o uso. Utilizar um *polystore* diminui drasticamente o problema de dividir os dados em diferentes SGBDs, mas ao mesmo tempo, traz com ele toda complexidade de uma implementação otimizada que comporta a flexibilidade de gerenciar vários sistemas distintos sem perder a integridade de dados.

A unidade de abstração no BigDAWG é chamada de *ilha de informação*. Cada ilha de informações tem uma linguagem de consulta, um modelo de dados e se conecta a um ou mais sistemas de armazenamento através do *shim*, o que por sua vez mapeia a linguagem da *ilha de informação* na consulta nativa do SGBDs correspondentes, como demonstra na figura 3. O *shim* possui um papel bastante similar com o *adaptador* do Blendb. BigDAWG oferece também consultas entre as *ilhas*, em outras palavras, quando uma consulta não pode ser traduzida para uma única consulta nativa de um determinado SGBD, é construído um *shim* que divide a consulta em sub consultas para então traduzir em buscas nativas. Atualmente, o BigDAWG suporta PostgreSQL, SciDB e Accumulo. (DUGGAN et al., 2015)

A proposta do BigDAWG é a que mais se aproxima do Blendb. Entretanto o BigDAWG também é uma ferramenta muito nova, mesmo que em estágio mais avançado que o Blendb, ela não cobre todos os SGBDs utilizados pelo C3SL. Pois como é um *polystore* genérico, não adota consultas analíticas utilizando métricas, dimensões e filtros como é o caso do Blendb (GADEPALLY et al., 2016).

FIGURA 3 – Arquitetura do Big-DAWG



FONTE: Retirado de Duggan et al. (2015, p. 5).

3.2 APACHE CALCITE

A proposta do Apache Calcite se define como um framework que oferece processamento e otimização de consultas. Sua abordagem é ser um gerenciador de dados dinâmico, contendo muitas das partes que compõem um sistema de gerenciamento de banco de dados convencional, mas tirando a parte de armazenar (BEGOLI et al., 2018).

No BigDAWG, cada *ilha de informações* tem uma linguagem de consulta, um modelo de dados e se conecta a um ou mais sistemas de armazenamento. A consulta entre mais de um SGBD é possível apenas dentro dos limites de uma única *ilha*. No Calcite, é construído uma abstração relacional única que permite a consulta em diferentes modelos de dados, em outras palavras, ela transforma a entrada SQL em álgebra relacional.

O *adapter* do Calcite é um padrão de arquitetura que define como o Calcite acessa as diversas fontes de dados. Essencialmente, um adaptador consiste em especificar as propriedades físicas da fonte de dados que está sendo acessada, além de manter um esquema que define os dados do modelo. Os dados em si são acessados fisicamente por meio de tabelas. O Calcite interage com as tabelas definidas no adapter para ler os dados conforme a consulta está sendo executada. O adaptador pode definir um conjunto de regras. Por exemplo, normalmente inclui regras para converter vários tipos de expressões relacionais lógicas nas expressões relacionais correspondentes. (BEGOLI et al., 2018)

O Apache Calcite visa substituir os componentes do SGBD acima do armazenamento. Já o Blendb traz uma proposta diferente, apenas construindo um nível acima dos SGBDs e realiza consultas analíticas nativas traduzidas pela API.

3.3 APACHE DRILL

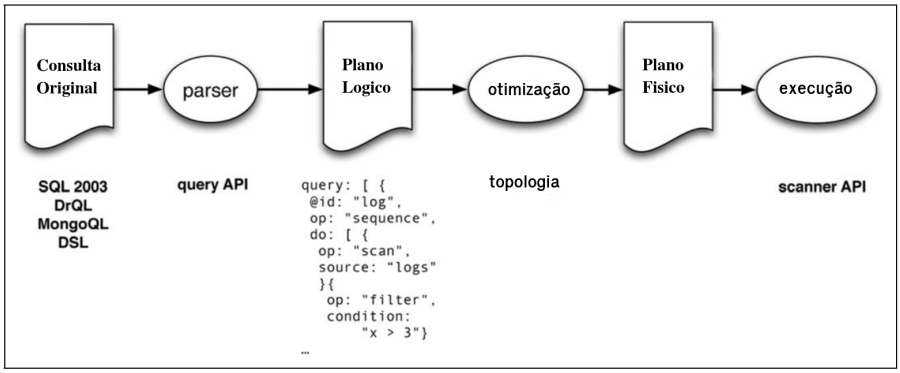
Além de diminuir a complexidade de consultas utilizando *polystores*, uma outra abordagem que trata da otimização do processamento de grande quantidade de dados, como o caso do Apache Drill.

Projetado para manipular até *petabytes* de dados em milhares de servidores. A proposta do Apache Drill é otimizar as consultas *ad-hoc*[1]. A arquitetura pode ser dividida em três níveis (HAUSENBLAS; NADEAU, 2013):

- **Usuário:** se refere às interfaces, como uma interface de linha de comandos (CLI) ou uma interface REST, para interação entre humanos e aplicativos.
- **Processamento:** permitindo linguagens de consulta, bem como planejador de consulta, execução e armazenamento.
- **Fontes de dados:** fontes de dados conectáveis, locais ou em *cluster*, fornecendo processamento de dados.

O Apache Drill não é um banco de dados, mas uma camada de consulta que funciona com várias fontes de dados integrados. A execução das buscas inicia transformando as consultas de linguagem sintáticas, como SQL, em uma forma interna conhecida como o *plano lógico*. Em seguida, o Apache Drill transforma o *plano lógico* em um *plano físico*, que é executado sobre as fontes de dados. A figura 4 apresenta o fluxo da consulta.

FIGURA 4 – Arquitetura do Apache Drill



FONTE: Retirado de Hausenblas e Nadeau (2013)

Vertentes diferente na pesquisa de *polystores*. Enquanto o Apache Drill aprofunda na otimização das consultas, o Blendb, atualmente, apenas traduz as consultas na API para as consultas nativas dos SGBDs integrados com o propósito de realizar pesquisas analíticas.

3.4 OUTRAS PROPOSTAS

Além dos *polystores* já apresentados nessa seção, existem diversos outros projetos na área de banco de dados federados. Podemos citar também o CloudMDSQL em Kolev, Bondiombouy e Valduriez (2016), o qual permite realizar consultas em diversos bancos de dados

[1] Consultas *ad-hoc*: consultas não generalizadas, ou seja, consultas utilizados para atender alguma demanda temporária.

heterogêneos, sendo eles relacionais ou NoSQL. Traduzindo uma única consulta em diversas consultas nativas, entretanto, a linguagem apresentada pelo CloudMDSQL é semelhante a SQL.

As abordagem dos projetos MISO (MultISstore Online) em LeFevre et al. (2014) e ESTOCADA em Bugiotti et al. (2016), focam em otimização dos *polystores*, seja em tradução da consulta ou na arquitetura dos bancos, mas nenhuma delas possui uma consulta analítica com métricas, dimensões e filtros como apresenta o Blendb. (LEFEVRE et al., 2014) (BUGIOTTI et al., 2016).

4 ADAPTADOR DE BANCO DE DADOS TEXTUAL PARA POLYSTORE ANALÍTICO

Tendo em vista as vantagens da construção de *polystores* para o cenário de grande quantidade de dados e com o projeto Blendb sendo uma das estratégias desenvolvidas dentro da UFPR, a proposta desse trabalho é adicionar mais um SGBD ao leque de adaptadores do Blendb, o adapter para o *Elasticsearch*.

4.1 MOTIVAÇÃO

O Elasticsearch traz um modelo diferente dos banco de dados tradicionais, apresentando uma perspectiva nova de trabalhar com os dados. Por se tratar de um modelo orientado a documento, com API RESTful, a integração com um mediador de bancos de dados analítico como é o Blendb se tornou facilitada.

De um lado, um banco de dados orientado a documentos já utilizado em diversos projetos do C3SL, e de outro, um projeto que visa implementar um *polystore* analítico. A escolha desta proposta se deve justamente por nos depararmos com duas partes que possuem grande potencial que podem ser integrados. A implementação de um adaptador Elasticsearch abre portas para a utilização do Blendb em todos os projetos armazenados em Elasticsearch, o que significa aumentar o potencial analítico sobre estes dados.

4.2 DESAFIOS

4.2.1 Inserção de dados no Elasticsearch

O Elasticsearch é um banco de dados que não trabalha com SQL e por isso, migrar os dados de um banco de dados tradicional para o Elasticsearch possui poucos caminhos, e o mais popular deles é através do *Logstash*, inserindo arquivos *CSVs*. A versão Logstash utilizada nesse trabalho é o 7.0.1, para sua utilização, é necessário um arquivo de configuração, demonstrado no código 4.2.1 com os seguintes campos:

- *input*: permite que uma fonte específica de eventos seja lida pelo Logstash, em outras palavras, configura o caminho do arquivo CSV que será lido.
  - *sicedb_path*: o Logstash acompanha a posição atual em cada arquivo gravando-o em um arquivo separado chamado sincedb, isso permite parar e reiniciar o Logstash e fazer com que ele continue da onde parou.
- *filter*: executa o processamento intermediário, geralmente são aplicados condicionalmente, dependendo das características do evento. Como por exemplo inserir o valor em

número ao invés de texto. Além de informar as colunas que serão lidas juntamente com a definição do separador no CSV.

- *output*: comunica com o Elasticsearch e define o *índice* que será armazenado os dados.

```
input {
   file {
      path => "/home/matriculas.csv"
   start_position => "beginning"
      sincedb_path => "/dev/null"
   }
}

filter {
   csv {
      separator => "|"
      columns => [
            "id_escola", "uf_estado", "nu_ano", "id_matricula",
            "id_aluno", "nu_idade", "id_turma"]

      convert => {
            "nu_ano" => "integer"
            "nu_idade" => "integer"}
   }
}

output {
   elasticsearch {
      hosts => "localhost:9200"
      index => "matriculas"
   }
   stdout {}
}
```

CÓDIGO 4.2.1: Exemplo de arquivo *conf* Logstash

4.2.2 Análise das consultas Elasticsearch

É importante ressaltar que *joins* não existem nas consultas Elasticsearch em sua forma nativa, como no SQL. Até mesmo a API de tradução de SQL[1] dentro do Elasticsearch não realiza traduções com *joins* (versão 6.4 do Elasticsearch).

A complexidade da conversão entre SQL e as consultas Elasticsearch torna o Blendb ainda mais atraente no sentido que, as requisições pela API do Blendb são totalmente baseadas

[1] *API SQL Translate*: aceita SQL em um documento JSON e traduz em consultas nativas do Elasticsearch.

em métricas, dimensões e filtros, desta maneira, facilita ao usuário e evita o caminho mais difícil de tentar adaptar as consultas SQL para o Elasticsearch.

O desafio agora é construir uma consulta Elasticsearch que englobe agregações, agrupamentos e filtros, de modo que seja possível realizar a tradução das métricas, dimensões e filtros em consulta Elasticsearch. O objetivo é encontrar um padrão nas consultas nativas do Elasticsearch, o qual auxiliem na etapa de construção do algoritmo de tradução no adaptador. Assim, a pergunta que utilizamos como exemplo foi: "média das idades e quantidade de matrículas, do sexo tipo 2, por estado", sobre a base de Dados Educacionais disponibilizados pelo C3SL.

Supondo que os documentos sejam um conjunto de dados com as mesmas chaves que o código 4.2.2. Tendo ciência que as chaves que iniciam com _ são criadas pelo próprio Elasticsearch.

```
{
  "_index": "matriculas",
  "_type": "_doc",
  "_id": "1",
  "_version": 1,
  "_score": 0,
  "_source": {
    "co_pessoa_fisica": "14467",
    "id_matricula": 2333,
    "idade": 14,
    "turma": "876",
    "municipio": "4202909",
    "estado": "42",
    "tp_sexo": "2"
  }
}
```

CÓDIGO 4.2.2: Exemplo de documento em Elasticsearch

A consulta que responderia a pergunta seria similar ao código 4.2.3. Dizemos similar, pois não há apenas uma forma para responder a pergunta, entretanto, a consulta demonstrada no código 4.2.3 é utilizada como base para a construção do adapter, posteriormente.

Algumas informações podem auxiliar na análise do código 4.2.3. Por exemplo, o primeiro *size*, cuja funcionalidade é apenas para configurar a quantidade de documentos consultados que irá aparecer no retorno, ao atribuir o valor 0, obtemos apenas os resultados das agregações em seus devidos agrupamentos e aplicado aos respectivos filtros. Respondendo a pergunta sem apresentar por quais documentos ele consultou. Já o *size* dentro do *group_by* serve para definir quantos grupos do agrupamento serão retornados, sendo assim, o valor 10000 é meramente um valor grande o suficiente para retornar todos os grupos, uma vez que

não há uma forma para dizer ao Elasticsearch que ele deve retornar todos os grupos. Por último, ao agrupar por uma *string*, é necessário adicionar *.keyword* após a chave (ELASTIC, 2019).

```
GET /matriculas/_search/
{
  "size": 0,
  "query": {
    "bool": {
       "must": [
          {"match": { "tp_sexo": 2 }}
       ]
    }
  },
  "aggs": {
    "group_by_estado": {
       "terms": {
          "field": "estado.keyword",
          "size": 10000
       },
       "aggs": {
          "average_idade": {
             "avg": { "field": "idade" }
          },
          "count_matriculas": {
             "value_count": { "field": "id_matricula" }
          }
       }
    }
  }
}
```

CÓDIGO 4.2.3: Exemplo de consulta nativa Elasticsearch

4.2.3 Análise do código Blendb

O projeto Blendb é desenvolvido em *TypeScript*, uma linguagem de programação que é compilada para JavaScript. O primeiro estudo foi feito sobre a árvore do repositório e identificado os arquivos que serão modificados na implementação. Além do próprio arquivo que será implementado o adaptador do Elasticsearch, é necessário conectar as partes, em uma ponta, a comunicação do adapter com a API, e por outro a conexão com o próprio Elasticsearch.

Uma das ferramentas facilitadoras do Typescript é o @type, uma grande vantagem do TypeScript, o qual permite que pacotes JavaScript sejam utilizados em programas TypeScript. Uma das tarefas que o @type facilitou no desenvolvimento é possibilitar a visualização dos parâmetros, retornos e estrutura das bibliotecas.

4.3 IMPLEMENTAÇÃO

O roteiro da implementação consistia em iniciar com um protótipo de adapter que realiza uma única consulta estática e definida dentro do algoritmo, a razão disso é para validar a consulta de modo que prove funcionar a comunicação com o Elasticsearch. Após receber um retorno positivo através do adapter, partimos para a construção da consulta dinamicamente.

O adaptador é uma classe abstrata do Blendb, e o adaptador do Elasticsearch é uma extensão concreta dessa classe, sendo assim, os métodos abaixo foram implementados.

4.3.1 *getQueryFromView()*

Os adaptadores do Blendb possuem várias funções em comum e algumas com implementações diferentes, uma delas é a *getQueryFromView()*. A tarefa principal do adaptador é traduzir a consulta requisitada pelo usuário na API do Blendb para a estrutura de consulta do SGBD respectivo, e é justamente essa funcionalidade que é implementada na função *getQueryFromView()*. A consulta na estrutura interna *view* formada pela API possui o esqueleto apresentado no apêndice A. É importante notar que as agregações, tipos dos dados e operadores são em forma numérica.

As consultas do Elasticsearch realizados pelo Blendb possuem um modelo base apresentado no código 4.3.1. O primeiro passo foi identificar os filtros na *view* e traduzí-lo em *must match* ou *must_not match*, consequentemente apenas as operações de igual ou diferente são aceitas no filtro para o adaptador Elasticsearch.

```
{
   "index": "matricula",
   "body": {
      "size": 0,
      "query": {
         "bool": {
            "must": [],
            "must_not": []
         }
      },
   "aggs": {}
   }
}
```

CÓDIGO 4.3.1: Estrutura base da consulta Elasticsearch

O pseudo código 4.3.2 apresenta a implementado para a função *getQueryFromView()*.

```
1:  function getQueryFromView(view)                    ▷ view possui tipo interno view
2:      inicia query
3:      filter = {}
4:      match = [ ]
5:      nMatch = [ ]
6:      if tamanho(view.clauses) > 0 then              ▷ view.clauses é um vetor
7:          for i = 0 : tamanho(view.clauses[0].filters) do
8:              if view.clauses[0].filters[i].operator == "! = " then
9:                  filter ← view.clauses[0].filters[i]      ▷ filter recebe um objeto
10:                 nMatch.push(filter)
11:             else
12:                 filter ← view.clauses[0].filters[i]
13:                 match.push(filter)
14:         Object.assign(query.query.bool, must : match)
15:         Object.assign(query.query.bool, must : match)
16:     partialQuery ← translateAggs(view, 0)
17:     Object.assign(query, partialQuery)
18:     return query
```

CÓDIGO 4.3.2: Pseudo código da função *getQueryFromView()*

A inicialização da variável *query* consiste em passar o objeto apresentado no código 4.3.1 para a variável. Em seguida, para cada filtro é criado um objeto contendo:

```
1   { "match": "dimensão" }
```

E adicionado à *must* para operações de igualdade ou *must_not* para operações de diferente.

4.3.1.1 Traduzir a agregação

Após os filtros, é construído as dimensões e métricas, respectivamente. Diferente da ordem dos filtros, as métricas precisam, necessariamente, ser construídas dentro das dimensões. Além disso, as dimensões precisam ser construídas uma dentro da outra, isto é, ao receber mais de uma dimensão, o Elasticsearch não realiza todas simultaneamente, é feito de forma hierárquica, como ilustra os *aggs* no código 4.3.3.

Para a traduzir as métricas e as dimensões, foi implementada uma função recursiva *translateAggs* que constrói primeiramente as métricas nas agregações mais internas e acrescentando as agregações exteriores. Demonstrado no código 4.3.4.

```
"aggs": {
   "group__by__estado": {
      "terms": {
         "field": "estado.keyword",
         "size": 10000
      },
      "aggs": {
         "group__by__sexo": {
            "terms": {
               "field": "tp__sexo.keyword",
               "size": 10000
            },
            "aggs": {
               "media__idade": {
                  "avg": {
                     "field": "idade"
                  }
               }
            }
         }
      }
   }
}
```

CÓDIGO 4.3.3: Estrutura base da consulta Elasticsearch

A função *translateAggs* executa pela primeira vez passando como parâmetro a *view* e o valor *zero*. A recursão vai adicionando *um* até que passe pela quantidade de dimensões que a *view* possui, esse incremento significa a quantidade de *aggs*[2] que a será construído para as dimensões. Na construção das métricas é construído para cada métrica uma função de agregação com a respectiva *chave* e todos colocado dentro de uma mesma *aggs*.

A parte das métricas é passado para a recursão anterior e então construído as dimensões em volta dela. Para cada dimensão passado pela API do Blendb, é construído um *aggs* que agrupa pela dimensão. Caso o tipo da dimensão não seja número, é adicionado **.keyword**.

Por fim, a recursão vai retornando e a última dimensão possui todas as outras dimensões e as métricas dentro. Retornando a consulta nativa completa para execução.

[2] *aggs*: definido por (GORMLEY, 2015) como um conjunto de valor de vários intervalos construídos dinamicamente

```
1: function translateAggs(view, numDimensions)
2:     partialQuery = {}
3:     if numDimensions == tamanho(view.dimensions) then
4:         for i = 0 : tamanho(view.metrics) do
5:             func ← view.metrics[i].aggregation
6:             aggrName ← view.metrics[i].name
7:             Object.assign(partialQuery, [aggrName] : [func] : aggrName)
8:         return partialQuery
9:     returnedQuery ← translateAggs(view, numDimensions + 1)
10:    if tamanho(view.dimensions) > 0 then
11:        groupby ← view.dimensions[numDimensions].name
12:        dim = view.dimensions[numDimensions].name
13:        if view.dimensions[numDimensions].dataType < 3 then
14:            dim ← dim                                  ▷ datatype: numbers
15:        else
16:            dim ← dim + ".keyword"                     ▷ datatype: strings
17:        Inicia aggregation
18:        Object.assign(aggregation[groupby], aggs : returnedQuery)
19:        return aggregation
```

CÓDIGO 4.3.4: Pseudo código da função *translateAggs()*

4.3.2 *executeQuery()*

Após construir dinamicamente a consulta para o Elasticsearch, é chamado a função que comunica com o SGBD e que realiza a consulta propriamente dita, a função *executeQuey*. Todos os adapters possuem uma função que realiza a busca, no caso do adapter Elasticsearch, possui a particularidade de precisar receber o *índice*, pois diferente nos casos dos SGBD relacionais, onde os atributos descritos no arquivo de configuração *.yaml* formam a estrutura do banco no Blendb e que possibilita os *joins*.

O Elasticsearch precisa de um único nome, o *índice*, para que ele saiba onde os documentos estão armazenados e realizar a consulta, entretanto esse índice precisa chegar de alguma forma até a função *executeQuery*. O Blendb, no momento, não possui uma forma própria para declarar, sendo assim, a maneira mais apropriada que encontramos para isso foi configurar no config.yaml e chegar até o *executeQuery* através da *view*.

4.3.2.1 *findIndex()*

No arquivo de configuração config.yaml é possível passar na sessão **views** uma combinação da opção **aliasAsName** com o **alias**, são eles, um booleano e o nome do índice, respectivamente. Ao passar o primeiro campo com o valor *true*, o alias é passado dentro da *view* na chave *name* quando a chave *origin* possui o valor booliano *true*, o que por sua vez significa que o índice está no *name* mais interno da *view*.

A função *findIndex* faz a busca do *origin* recursivamente até encontrar valor *true*, e ao encontrar, retorna o *name* da instância.

4.3.3 *Resultado da busca*

Exatamente nesse ponto do adaptador, a tradução das consultas para Elasticsearch estão completas. O retorno da busca no Elasticsearch é um JSON extremamente confuso, pois além dos resultados, é retornado também as chaves criadas pelo próprio Elasticsearch (iniciadas com "_"), o que são úteis caso o propósito seja analisar desempenho além de consultar os dados, como demonstrado no código 4.3.6.

Além da grande quantidade de chaves que o Elasticsearch retorna, é importante notar que os valores são retornados dentro de *buckets*. *Bucket* é associado a um critério (dependendo do tipo de agregação) que determina se um documento no contexto se insere ou não nele. Em outras palavras, os *buckets* definem um conjuntos de documentos.

4.3.3.1 *Formatar retorno*

Sabendo o comportamento do retorno do Elasticsearch, notamos que cada dimensão do Blendb constrói um *buckets* no retorno, e a combinação de dimensões são combinações de *buckets*. O último passo agora é formatar o retorno para se igualar com os outros adaptadores. O formato padrão do retorno do Blendb é um vetor com objetos que possui a métrica com o valor resultante, e as suas respectivas dimensões.

Para formatar o retorno, foi implementando uma função recursiva que itera dentro dos *buckets*, e cada iteração buscar a chave *key* até que encontre a métrica e então constrói um vetor similar ao código 4.3.5.

```
[
  {
    "idade": 11.93572543837987,
    "tp_sexo": "1"
  },
  {
    "idade": 12.097880085419062,
    "tp_sexo": "2"
  }
]
```

CÓDIGO 4.3.5: Retorno da consulta Blendb

```
[
  {
    "took": 371,
    "timed_out": false,
    "_shards": {
      "total": 1,
      "successful": 1,
      "skipped": 0,
      "failed": 0
    },
    "hits": {
      "total": {
        "value": 10000,
        "relation": "gte"
      },
      "max_score": null,
      "hits": [

      ]
    },
    "aggregations": {
      "tp_sexo": {
        "doc_count_error_upper_bound": 0,
        "sum_other_doc_count": 0,
        "buckets": [
          {
            "key": "1",
            "doc_count": 2032826,
            "idade": {
              "value": 11.93572543837987
            }
          },
          {
            "key": "2",
            "doc_count": 1921351,
            "idade": {
              "value": 12.097880085419062
            }
          }
        ]
      }
    }
  }
]
```


CÓDIGO 4.3.6: Retorno da consulta Elasticsearch

No caso particular em que o filtro possui uma dimensão não passada explicitamente na consulta Blendb, o adaptador não conseguirá adicioná-lo no objeto do vetor retorno, a razão se deve ao Elasticsearch, uma vez que o filtro se comporta individualmente das métricas e dimensões na hora da consulta. O Elasticsearch entende que ao filtrar os documentos, o retorno é claramente restrito respeitando o filtro. De forma mais visual, caso a busca do Blendb seja:

```
localhost:3000/v1/data?metrics=idade&dimensions=estado&filters=tp_
↪  sexo==2,estado==41
```

O vetor resultado gerado pelo adapter fica como no código 4.3.7. A métrica está presente sempre, já a dimensão que aparece é apenas o que foi explicitado na API do Blendb, caso contrário, mesmo que seja utilizado nos filtros, não será retornado no vetor resultante gerado.

```
[ { "idade": 12.222776617437033,
   "estado": "41" } ]
```

CÓDIGO 4.3.7: Retorno do adapter sem dimensão e com filtro

O pseudo código 4.3.8 apresenta mais detalhadamente a construção do resultado da consulta. O algoritmo inicialmente recebe como parâmetro a *view*, o resultado do próprio Elasticsearch e o valor *zero*. A *view* é necessário para encontrar o nome das dimensões. Como a ordem dos agrupamentos são mantidos desde a construção da consulta até o retorno do Elasticsearch, é possível para cada *bucket* encontrar *chave* da dimensão pela *view*.

Cada iteração percorre um *bucket*, um vetor, que possui os valores de um agrupamento, construindo um objeto com os valores das dimensões e por último, as métricas são adicionadas nos objetos que correspondem. Pois durante a construção do vetor resultado, os objetos dentro, possuem apenas os *valores*, isto é, é necessário primeiro construir o vetor com objetos:

```
[ { idade: { value: 12.222776617437033 },
   key: "41" } ]
```

Para que então sejam transformadas assim:

```
[ { "idade": 12.222776617437033,
   "estado": "41" } ]
```

```
1: function formateResult(view, result, numDimensions)
2:     back = [ ]
3:     resultArray = [ ]
4:     if numDimensions == 0 and tamanho(view.dimensions) > 0 then
5:         firstDim ← view.dimensions[0].name
6:         for i = 0 : tamanho(result[0].aggregations[firstDim].buckets) do
7:             back ← formateResult(view, result.aggregations[firstDim].buckets[i], 1)
8:             for j = 0 : tamanho(back) do
9:                 dimKey ← result[0].aggregations[firstDim].buckets[i].key
10:                Object.assign(back[j], [firstDim] : dimKey)
11:            resultArray ← resultArray.concat(back)
12:        return resultArray
13:    if numDimensions == tamanho(view.dimensions) then
14:        metricResult = {}
15:        findPrefix ← result[0]
16:        if tamanho(view.dimensions) == 0 then
17:            findPrefix ← result[0].aggregations
18:        for i = 0 : tamanho(view.metrics do
19:            met ← view.metrics[i].name
20:            Object.assign(metricsResult, [met] : findPrefix[met].value)
21:        return metricsResult
22:    nDim ← view.dimensions[numDimensions].name
23:    for i = 0 : tamanho(result[0][nDim].buckets) do
24:        aux ← result[0][nDim].buckets[i]
25:        back ← formateResult(view, aux, numDimensions + 1)
26:        for j = 0 : tamanho(back) do
27:            Object.assign(back[j], [nDim] : result[0][nDim].buckets[i].key)
28:        resultArray ← resultArray.concat(back)
29:    return resultArray
```

CÓDIGO 4.3.8: Pseudo código da função *formateResult()*

5 RESULTADOS EXPERIMENTAIS

Para realizar a validação do adapter, utilizamos duas bases. A primeira para verificar a veracidade da busca em combinação com as funções de agregação. Ela possui apenas 10 documentos com a estrutura do código 5.0.1.

```
PUT teste/_doc/1
{
   "matricula" : 11111,
   "sexo" : "2",
   "estado" : "41",
   "idade" : 13,
   "ano" : 2019,
   "nome": "ana maria"
}
```

CÓDIGO 5.0.1: Estrutura da base teste

O propósito de testar uma base pequena é verificar as funções de agregação ao agrupar. Quando a base é menor, é possível fazer o cálculo e validar manualmente os resultados. Por exemplo: verificar a média das idades de 10 documentos é mais fácil do que em 1 milhão de documentos.

Após validar manualmente os resultados na base teste, utilizamos os Microdados do Censo Escolar. Coordenado pelo Instituto Nacional de Estudos e Pesquisas (Inep) e realizado em regime de colaboração entre as secretarias estaduais e municipais de educação e com a participação de todas as escolas públicas e privadas do país. É feito uma coleta de informações da educação básica e considerada como a mais importante pesquisa estatística educacional brasileira (INEP, 2018).

Por conta da quantidade de dados, retiramos apenas parte do microdados. O escolhido é o Microdados do Censo Escolar da Região Sul, pois somente essa fração do dado já obtemos 7.112.070 documentos. O suficiente para realizar os testes.

5.1 ARQUIVO DE CONFIGURAÇÃO

Como apresentado anteriormente, o Blendb necessita do arquivo de configuração *.yaml* para poder realizar a busca. Sendo assim, para a base do Censo Escolar cuja a estrutura é apresentada nos códigos 5.1.1 e 5.1.2, a sua estrutura completa se encontra no apêndice B. Na prática, os códigos são apenas componentes do documento, a divisão feita aqui foi para facilitar a análise dos campos. O código 5.1.1 apresenta apenas os campos que utilizamos como métricas

no Blendb, sejo assim, para cada campo apresentado no código 5.1.1 possui uma função de agregação atribuído.

```
PUT matricula_sul/_doc/1
{
   "NU_DIAS_ATIVIDADE": 5,
   "NU_DURACAO_TURMA": 275,
   "NU_IDADE": 5,
   "ID_MATRICULA": 198145744,
   "ID_TURMA": "1025084",
   "CO_ENTIDADE": "41135946",
   "CO_PESSOA_FISICA": "128066622812",
```

CÓDIGO 5.1.1: Estrutura da base Microdados do Censo Escolar

O código 5.1.2 descreve os campos utilizados para as dimensões e filtros. A escolha depende de quais campos serão utilizados para fazer agrupamento ou filtrar.

```
   "NU_ANO": 2012,
   "TP_NACIONALIDADE": "1",
   "IN_PROFISSIONALIZANTE": "0",
   "TP_SEXO": "1",
   "CO_UF": "41",
   "CO_UF_NASC": "41",
   "NU_MES": 12,
   "TP_COR_RACA": "1",
   "IN_NECESSIDADE_ESPECIAL": "0",
   "NU_DIA": 29,
   "IN_REGULAR": "1",
```

CÓDIGO 5.1.2: Estrutura da base Microdados do Censo Escolar

Além dos dados, é necessário o arquivo de configuração que defina as métricas e dimensões. O código 5.1.3 apresenta uma abordagem possível. O documento *.yaml* que cobre os códigos 5.1.1 e 5.1.2 se encontra no apêndice C. A diferença das métricas e das dimensões está na função de agregação. Já os filtros são definidos também na sessão *dimensions*.

```yaml
metrics:
   links: [ ]
   obj:
      -
         name: "NU_IDADE"
         dataType: "integer"
         aggregation: "avg"
         description: "Média das idades"
      -
         name: "ID_MATRICULA"
         dataType: "integer"
         aggregation: "count"
         description: "Número de matrículas"
      -
         name: "ID_TURMA"
         dataType: "integer"
         aggregation: "count"
         description: "Número de turmas"
dimensions:
   links: [ ]
   obj:
      -
         name: "NU_ANO"
         dataType: "integer"
         description: "Ano da matrícula"
      -
         name: "TP_NACIONALIDADE"
         dataType: "string"
         description: "Tipo de nacionalidade"
      -
         name: "TP_SEXO"
         dataType: "string"
         description: "Sexo do aluno"
      -
         name: "CO_UF"
         dataType: "string"
         description: "Estado"
```

CÓDIGO 5.1.3: Estrutura das métricas no arquivo *.yaml*

5.2 BASE TESTE

A base teste possui 10 documentos. Os testes aplicados sobre eles são de agrupamento e validar a média das idades. A tabela 3 apresentamos os dez documentos. Após inserir o documento, um novo campo "_id"é adicionado automaticamente. A tabela 3 abaixo não representa o modelo armazenado no Elasticsearch, pois não é um SGBD relacional.

matricula	sexo	estado	idade	ano	nome	_id
11111	“2”	“41”	13	2019	“ana maria”	1
22222	“2”	“41”	17	2019	“joana”	2
33333	“2”	“41”	15	2019	“maria”	3
44444	“1”	“41”	19	2019	“mateus”	4
55555	“1”	“41”	20	2019	“joao”	5
66666	“1”	“42”	13	2019	“marcos”	6
77777	“2”	“42”	12	2019	“carla”	7
88888	“2”	“42”	22	2019	“valentina”	8
99999	“2”	“43”	16	2019	“lua”	9
11112	“2”	“43”	30	2019	“bruna”	10

TABELA 3 – Base teste

A pergunta que fizemos foi, qual o “número de matrículas, do sexo tipo 2, por estado”. Traduzido para a consulta Blendb fica:

```
localhost:3000/v1/data
↪  ?metrics=matricula&dimensions=estado&filters=sexo==2
```

O adapter do Elasticsearch irá construir uma consulta nativa do SGBD. A consulta construída é apresentado no código 5.2.1.

```
index: 'teste',
body: {
  "size": 0,
  "query": {
    "bool": {
       "must": [ {"match": { "sexo": 2 }} ],
       "must_not": []
    }
  },
  "aggs": {
    "estado": {
       "terms": {
          "field": "estado.keyword",
          "size": 100000
       },
       "aggs": {
          "matricula": { "value_count": { "field": "matricula" } }
 } } } }
```

CÓDIGO 5.2.1: Consulta construída pelo adaptador Elasticsearch

Ao realizar a consulta no Elasticsearch, o SGBD retorna um grande JSON, o qual é apresentado no apêndice D. E após formatado para o padrão Blendb, obtemos o código 5.2.2. Os valores retornados conferem com os valores calculados manualmente.

```
[ { "matricula": 3,
    "estado": "41"
  },
  { "matricula": 2,
    "estado": "42"
  },
  { "matricula": 2,
    "estado": "43"
} ]
```

CÓDIGO 5.2.2: Resultado formatado da consulta na base teste

5.3 BASE MICRODADOS DO CENSO ESCOLAR DA REGIÃO SUL

Ao realizar a mesma pergunta, qual o "número de matrículas, do sexo tipo 2, por estado", na base Microdados do Censo Escolar da Região Sul. A consulta Blendb fica:

```
localhost:3000/v1/data?metrics=ID_MATRICULA&dimensions=CO_
↪  UF&filters=TP_SEXO==2
```

O adapter do Elasticsearch irá construir uma consulta nativa similar ao código 5.2.1 apresentado na base teste. E o retorno formatado da consulta é apresentado no código 5.3.1.

```
[ { "ID_MATRICULA": 1358024,
    "CO_UF": "41"
  },
  { "ID_MATRICULA": 1283225,
    "CO_UF": "43"
  },
  { "ID_MATRICULA": 818273,
    "CO_UF": "42"
} ]
```

CÓDIGO 5.3.1: Resultado formatado da consulta na base do Censo Escolar

Por fim, realizamos alguns testes como por exemplo, consulta com duas métricas, duas dimensões e dois filtros. A consulta feita sobre o Blendb foi:

```
http://localhost:3000/v1/data?metrics=NU_IDADE,ID_
↪ MATRICULA&dimensions=CO_UF,TP_SEXO&filters=TP_
↪ NACIONALIDADE==1,CO_UF!=41
```

O adapter funcionou sem nenhuma falha, retornando o código 5.3.2.

```
[ { "NU_IDADE": 12.14378191488965,
    "ID_MATRICULA": 1340871,
    "TP_SEXO": "1",
    "CO_UF": "43"
  },
  { "NU_IDADE": 12.486820611733604,
    "ID_MATRICULA": 1280067,
    "TP_SEXO": "2",
    "CO_UF": "43"
  },
  { "NU_IDADE": 11.38062870138416,
    "ID_MATRICULA": 866230,
    "TP_SEXO": "1",
    "CO_UF": "42"
  },
  { "NU_IDADE": 11.43296155234524,
    "ID_MATRICULA": 814458,
    "TP_SEXO": "2",
    "CO_UF": "42"
} ]
```

CÓDIGO 5.3.2: Resultado formatado da consulta com duas métricas e dimensões

6 CONCLUSÃO

O presente trabalho teve como objetivo demonstrar o processo da construção do adapter Blendb para o Elasticsearch. A proposta permite uma forma mais simplificada do usuário realizar consultas Elasticsearch. Um novo adapter significa agregar um novo sistema de banco de dados ao leque de SGBDs suportados pelo Blendb. Atualmente, em produção, apenas bancos relacionais são utilizados, agregar um banco orientado à documento como o Elasticsearch, aumenta o horizonte do Blendb no sentido de mostrar os pontos que possam deixar a comunicação da API com o adapter mais completa.

O estudo realizado abrange também as dificuldades encontradas para construir uma consulta baseada em JSON, como é o caso do Elasticsearch. O modelo relacional é consultado, em sua maioria, via SQL, o seu comportamento baseia se em tabelas diferente contendo parte das informações e construída novas tabelas através de *joins* de acordo com o comando SQL. O papel do *join* é extremamente importante para que os dados consultados sejam efetivamente buscados apropriadamente. Entretanto em modelos orientados à documento possuem apenas *index*, *joins* não são essenciais.

Mesmo que o Elasticsearch provenha uma API chamada de *SQL Translate API*, oferecendo a possibilidade de realizar consultas diretamente em SQL, o comando *join* não está implementado. Ao armazenar os dados no modelo orientado à documento, é menos comum que os dados sejam divididos igual como acontece no modelo relacional, a ponto de realizar normalização de banco de dados para construir sua estrutura. Portanto, o adapter Elasticsearch realiza a tradução de uma consulta Blendb direcionado para um banco de dados Elasticsearch, o que não significa realizar uma tradução de consultas SQL para consultas Elasticsearch ou vice-versa.

Como a linguagem do Blendb é formada por métricas, dimensões e filtros, a construção da consulta Elasticsearch baseado na consulta Blendb não possui grandes incompatibilidades. Além do mais, por se tratar de um banco orientado à documento, temos uma grande vantagem, não tem a necessidade de construir o grafo da estrutura do banco contendo as colunas de cada tabela, para que o Blendb saiba como construir os *joins*. Uma vez que não realiza *joins*, a tradução é mais simples, pois os campos passados na API do Blendb já podem utilizar direto na consulta Elasticsearch.

O maior empecilho encontrado é referente aos filtros. Nas buscas Elasticsearch, as dimensões podem ser divididas em textos ou números, adicionando *.keyword* ou não na chave. Já os filtros são traduzidos diretamente em *must match* ou *must_not match*, o que significa a não possibilidade de realizar operações diferentes de igualdade ou diferente nos filtros na API do Blendb, o uso delas será interpretado como operador de igualdade.

Os trabalhos futuros caminham paralelamente aos outros SGBDs já em produção do Blendb. Atualmente, o Blendb permite apenas consultas, mas o objetivo é se tornar um *polystore* completo, em outras palavras, enquanto o Blendb não se tornar um *polystore*, o adapter Elasticsearch terá que se aprimorar, incluindo novas funcionalidades, para que acompanhe o progresso do Blendb.

REFERÊNCIAS

AMAZON. **Documentação da AWS**. Disponível em: <https://docs.aws.amazon.com/index.html>. Acesso em: 3 jun. 2019.

BEGOLI, Edmon et al. Apache Calcite: A Foundational Framework for Optimized Query Processing Over Heterogeneous Data Sources. ACM, 2018. DOI: 10.1145/3183713.3190662.

BUGIOTTI, Francesca et al. Flexible Hybrid Stores: Constraint-Based Rewriting to the Rescue. **2016 IEEE 32nd International Conference on Data Engineering (ICDE)**, IEEE, Helsinki, Finland, 2016. DOI: 10.1109/ICDE.2016.7498353.

C3SL. **Documentação do Blendb**. Disponível em: <https://gitlab.c3sl.ufpr.br/c3sl/blendb/wikis/home>. Acesso em: 3 abr. 2019.

DATE, Christopher J. **Introdução a sistemas de bancos de dados 8ª Edição**. Rio de Janeiro: Elsevier, 2003. 803 p.

DUGGAN, Jennie et al. The BigDAWG polystore system. **ACM Sigmod Record**, ACM, v. 44, n. 2, 2015.

ELASTIC. **Documentação do Elasticsearch**. Disponível em: <https://www.elastic.co/guide/index.html>. Acesso em: 16 jun. 2019.

GADEPALLY, Vijay et al. The BigDAWG polystore system and architecture. **High Performance Extreme Computing Conference (HPEC)**, IEEE, Waltham, MA, USA, 2016. DOI: 10.1109/HPEC.2016.7761636.

GORMLEY Clinton e Tong, Zachary. **Elasticsearch: The Definitive Guide**: A Distributed Real-Time Search and Analytics Engine. [S.l.]: O'Reilly Media, 2015. 684 p.

HAUSENBLAS, Michael; NADEAU, Jacques. Apache Drill: Interactive Ad-Hoc Analysis at Scale, 2013. DOI: 10.1089/big.2013.0011.

INEP. **Página dos Microdados disponíveis**. Disponível em: <http://portal.inep.gov.br/web/guest/dados>. Acesso em: 10 jun. 2018.

KOLEV, Boyan; BONDIOMBOUY, Carlyna; VALDURIEZ, Patrick. The CloudMdsQL Multistore System. **ACM Sigmod Record**, ACM, 2016. DOI: 10.1145/2882903.2899400.

KORTH Henry F. e Silberschatz, Abraham. **Database System Concepts**. [S.l.]: McGraw-Hill, 1991. 694 p.

LEFEVRE, Jeff et al. MISO: Souping Up Big Data Query Processing with a Multistore System. ACM New York, 2014. DOI: 10.1145/2588555.2588568.

RAMAKRISHNAN Raghu e Gehrke, Johannes. **Sistemas De Gerenciamento De Bancos De Dados**. Brasil: AMGH, 2007. 912 p.

STONEBRAKER Michael e Çetintemel, Ugur. “One size fits all”: an idea whose time has come and gone. **21st International Conference on Data Engineering (ICDE’05)**, IEEE, Washington, DC, USA, 2005. DOI: 10.1109/ICDE.2005.1.

APÊNDICES

APÊNDICE A – ESTRUTURA DA *VIEW* BLENDB


```
{
   "metrics": [{
     "name": "idade",
     "aggregation": 2
   }],
   "dimensions": [{
     "name": "tp_sexo",
     "dataType": 3
   }],
   "clauses": [{
      "filters": [{
        "target": {
          "name": "tp_sexo"
        },
        "operator": 1,
        "value": "2"
      }]
   }]
   "origin": false,
   "operation": {
      "opcode": 2,
      "values": [{
         "metrics": [{
            "name": "idade",
            "aggregation": 4,
            "description": "Média das idades"
         }],
         "dimensions": [{
            "name": "estado",
            "dataType": 3
         }, {
            "name": "tp_sexo",
            "dataType": 3
         }],
      "clauses": [],
      "origin": true,
```

```
        "operation": {
          "opcode": 0,
          "values": []
        },
        "name": "\"matriculas\""
        }]
  },
  "name": "view_0937118672bc0e72eeaa51772be3967584d55e37"
}
```

APÊNDICE B – EXEMPLO DE DOCUMENTO DO MICRODADOS CENSO ESCOLAR

```
PUT matricula_sul/_doc/1
{
   "NU_DIAS_ATIVIDADE": 5,
   "NU_DURACAO_TURMA": 275,
   "NU_IDADE": 5,
   "ID_MATRICULA": 198145744,
   "ID_TURMA": "1025084",
   "CO_ENTIDADE": "41135946",
   "CO_PESSOA_FISICA": "128066622812",
   "NU_ANO": 2012,
   "TP_NACIONALIDADE": "1",
   "IN_PROFISSIONALIZANTE": "0",
   "TP_SEXO": "1",
   "CO_UF": "41",
   "CO_UF_NASC": "41",
   "NU_MES": 12,
   "TP_COR_RACA": "1",
   "IN_NECESSIDADE_ESPECIAL": "0",
   "NU_DIA": 29,
   "IN_REGULAR": "1",
   "TP_LOCALIZACAO": "1",
   "IN_TRANSP_EMBAR_35": null,
   "IN_TRANSP_BICICLETA": null,
   "IN_TRANSP_OUTRO_VEICULO": null,
   "IN_TRANSP_TR_ANIMAL": null,
   "IN_DEF_MULTIPLA": null,
   "TP_REGULAMENTACAO": "1",
   "IN_TRANSP_TREM_METRO": null,
   "IN_RECURSO_AMPLIADA_20": null,
   "IN_DEF_FISICA": null,
   "CO_MUNICIPIO_END": "4119152",
   "IN_TRANSPORTE_PUBLICO": "0",
   "IN_SUPERDOTACAO": null,
   "IN_TRANSP_EMBAR_ATE5": null,
   "IN_RECURSO_AMPLIADA_16": null,
```

"IN_RECURSO_BRAILLE": null,
"CO_MICRORREGIAO": "41037",
"IN_DEF_INTELECTUAL": null,
"TP_LOCALIZACAO_DIFERENCIADA": "0",
"IN_TRANSP_EMBAR_5A15": null,
"TP_ZONA_RESIDENCIAL": "1",
"TP_MEDIACAO_DIDATICO_PEDAGO": "1",
"TP_CATEGORIA_ESCOLA_PRIVADA": "1",
"CO_CURSO_EDUC_PROFISSIONAL": null,
"NU_DUR_ATIV_COMP_OUTRAS_REDES": 0,
"IN_MANT_ESCOLA_PRIVADA_SIND": "0",
"NU_DUR_AEE_MESMA_REDE": 0,
"IN_RECURSO_LABIAL": null,
"IN_MANT_ESCOLA_PRIVADA_S_FINS": "1",
"TP_ETAPA_ENSINO": "2",
"IN_TRANSP_VANS_KOMBI": null,
"IN_TRANSP_EMBAR_15A35": null,
"TP_DEPENDENCIA": "4",
"TP_UNIFICADA": "0",
"IN_MANT_ESCOLA_PRIVADA_SIST_S": "0",
"IN_AUTISMO": null,
"IN_TRANSP_MICRO_ONIBUS": null,
"NU_DUR_ATIV_COMP_MESMA_REDE": 0,
"IN_RECURSO_AMPLIADA_24": null,
"IN_RECURSO_NENHUM": null,
"IN_MANT_ESCOLA_PRIVADA_EMP": "0",
"IN_MANT_ESCOLA_PRIVADA_ONG": "0",
"CO_REGIAO": "4",
"IN_RECURSO_LEDOR": null,
"CO_DISTRITO": "411915205",
"CO_PAIS_ORIGEM": "76",
"IN_SINDROME_ASPERGER": null,
"TP_CONVENIO_PODER_PUBLICO": null,
"IN_SINDROME_RETT": null,
"TP_RESPONSAVEL_TRANSPORTE": null,
"NU_IDADE_REFERENCIA": 4,
"IN_TRANSP_ONIBUS": null,
"IN_BAIXA_VISAO": null,

```
    "TP_TIPO_TURMA": "0",
    "CO_MESORREGIAO": "4110",
    "NU_ANO_CENSO": 2017,
    "IN_RECURSO_TRANSCRICAO": null,
    "IN_SURDEZ": null,
    "CO_MUNICIPIO": "4119152",
    "IN_EDUCACAO_INDIGENA": "0",
    "CO_UF_END": "41",
    "IN_SURDOCEGUEIRA": null,
    "IN_CEGUEIRA": null,
    "TP_INGRESSO_FEDERAIS": null,
    "IN_EJA": "0",
    "IN_CONVENIADA_PP": "0",
    "IN_RECURSO_LIBRAS": null,
    "IN_RECURSO_INTERPRETE": null,
    "CO_MUNICIPIO_NASC": "4106902",
    "IN_ESPECIAL_EXCLUSIVA": "0",
    "IN_TRANSTORNO_DI": null,
    "TP_OUTRO_LOCAL_AULA": "3",
    "NU_DUR_AEE_OUTRAS_REDES": 0,
    "IN_DEF_AUDITIVA": null
}
```

APÊNDICE C – EXEMPLO DO ARQUIVO DE CONFIGURAÇÃO

```
views:
  links: []
  obj:
    -
      alias: matriculas
      aliasAsName: true
      origin: true
      dimensions:
        - "NU_ANO"
        - "NU_ANO_CENSO"
        - "TP_NACIONALIDADE"
        - "IN_PROFISSIONALIZANTE"
        - "TP_SEXO"
        - "CO_UF"
        - "CO_UF_NASC"
        - "NU_MES"
        - "TP_COR_RACA"
        - "IN_NECESSIDADE_ESPECIAL"
        - "NU_DIA"
        - "IN_REGULAR"
      metrics:
        - "NU_DIAS_ATIVIDADE"
        - "NU_DURACAO_TURMA"
        - "NU_IDADE"
        - "ID_MATRICULA"
        - "ID_TURMA"
        - "CO_ENTIDADE"
        - "CO_PESSOA_FISICA"
metrics:
  links: []
  obj:
    -
      name: "NU_DIAS_ATIVIDADE"
      dataType: "integer"
      aggregation: "avg"
      description: "Média dos dias de atividade"
```

```
    -
      name: "NU_DURACAO_TURMA"
      dataType: "integer"
      aggregation: "avg"
      description: "Média da duração das turmas"
    -
      name: "NU_IDADE"
      dataType: "integer"
      aggregation: "avg"
      description: "Média das idades"
    -
      name: "ID_MATRICULA"
      dataType: "integer"
      aggregation: "count"
      description: "Número de matrículas"
    -
      name: "ID_TURMA"
      dataType: "integer"
      aggregation: "count"
      description: "Número de turmas"
    -
      name: "CO_ENTIDADE"
      dataType: "integer"
      aggregation: "count"
      description: "Número de entidades"
    -
      name: "CO_PESSOA_FISICA"
      dataType: "integer"
      aggregation: "count"
      description: "Número de pessoas físicas"
dimensions:
  links: []
  obj:
    -
      name: "NU_ANO"
      dataType: "integer"
      description: "Ano da matrícula"
    -
      name: "NU_ANO_CENSO"
```

```
    dataType: "integer"
    description: "Ano do censo"
-
    name: "TP_NACIONALIDADE"
    dataType: "string"
    description: "Tipo de nacionalidade"
-
    name: "IN_PROFISSIONALIZANTE"
    dataType: "string"
    description: "Matrícula do tipo profissionalizante"
-
    name: "TP_SEXO"
    dataType: "string"
    description: "Sexo do aluno"
-
    name: "CO_UF"
    dataType: "string"
    description: "Estado"
-
    name: "CO_UF_NASC"
    dataType: "string"
    description: "Estado de nascimento"
-
    name: "NU_MES"
    dataType: "string"
    description: "Mês"
-
    name: "TP_COR_RACA"
    dataType: "string"
    description: "Cor da raça"
-
    name: "IN_NECESSIDADE_ESPECIAL"
    dataType: "string"
    description: "Aluno de necessidade especial"
-
    name: "NU_DIA"
    dataType: "string"
    description: "Dia"
-
```

```
            name: "IN_REGULAR"
            dataType: "string"
            description: "Matrícula do tipo regular"
enumTypes:
    links: []
    obj: []
sources:
    links: []
    obj: []
```

APÊNDICE D – RETORNO DA CONSULTA ELASTICSEARCH ANTES DA FORMATAÇÃO


```
{
  "took": 1,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 7,
      "relation": "eq"
    },
    "max_score": null,
    "hits": []
  },
  "aggregations": {
    "estado": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
      "buckets": [
        {
          "key": "41",
          "doc_count": 3,
          "matricula": {
            "value": 3
          }
        },
        {
          "key": "42",
          "doc_count": 2,
          "matricula": {
            "value": 2
          }
        },
```

```
      {
        "key": "43",
        "doc_count": 2,
        "matricula": {
          "value": 2
        }
      }
    ]
  }
 }
}
```