BIBINSON CELISSAINT E LUIZ EDUARDO CAVALHEIRO

ESTUDO DE INTEGRAÇÃO DE DADOS ABERTOS USANDO APACHE DRILL.

(*versão pré-defesa, compilada em 10 de janeiro de 2022*)

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência Da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Marcos Didonet Del Fabro.

CURITIBA PR

2021

RESUMO

O crescimento de dados abertos a cada ano geram inúmeras possibilidades de análises que podem trazer resultados interessantes tanto para empresas quanto para a sociedade. Hoje uma grande quantidade de dados está disponível para ser utilizada por quem tiver interesse. Existem inúmeras dificuldades em relação ao uso deste dados, como por exemplo a inexistência de um padrão e falta de documentação acessível para o entendimento dos dados e de seus possíveis mapeamentos com outros conjuntos. Sendo assim, para que análises envolvendo diversos conjuntos de dados ao mesmo tempo sejam feitas, é necessário uma forma de utilização que possibilite o uso integrado destes dados diversos garantindo o correto mapeamento entre eles. Diversas são as técnicas de integração de dados, cujo objetivo é unir conjuntos de dados diversos através de alguma relação, realizando mapeamentos que possibilitem a correlação entre bases de dados diversas. Técnicas e estudos nesta área de integração de dados se tornam mais relevantes para o contexto de dados abertos. O presente trabalho propõe uma abordagem que possibilita o acesso entre estes conjuntos de dados, sendo implementado utilizando a API do Apache Drill, que possibilita o uso de conjunto de dados com formatos e esquemas heterogêneos. A arquitetura utilizada tem como objetivo possibilitar principalmente a experimentação e teste de soluções existentes de integração de dados. Com isso, foi possível verificar a utilidade destes formatos para disponibilização de dados abertos. Além da implementação e dos resultados obtidos, também mostramos os desafios da área de integração de dados para dados abertos, assim como sua importância.

Palavras-chave: Integração de Dados, Apache Drill, Dados Abertos.

ABSTRACT

The increase of open data every year generates countless possibilities of analysis that can bring interesting results for enterprises and society. Today a big amount of data is available for those who are interested in using them. There are countless difficulties regarding the use of this data, for example the inexistence of a common patern and accessible documentation for understanding the data and possible mappings with other datasets. Therefore, for performing analysis involving several datasets simultaneously, it is necessary to have a way of usage that allows the integrated use of these datasets, for ensuring the correct mapping between them. There are many data integration techniques, whose objectives are to unite several data set through some relation, creating mappings that allow the correlation between them. Techniques and studies in data integration area become more relevant in the context of open data. The present work proposes an approach that allows access between these datasets, and is implemented using the Apache Drill API, which allows the use of datasets with heterogeneous formats and schemas. The architecture used aims to allow mainly the experimentation and test of existent data integration solutions. With that, we are able to verify the utility of these formats for open data availability. Besides the implementation and results obtained, we also show the challenges of data integration area for open data, as well as its importance.

Keywords: Data Integration, Apache Drill, Open Data.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

CSV	Comma Separated Values
JSON	JavaScript Object Notation
ETL	Extraction Transformation Load
URL	Uniform Resource Locator
DINF	Departamento de Informática
UFPR	Universidade Federal do Paraná
SGBD	Sistema Gerenciador de Banco de Dados
SQL	Structured Query Language
API	Application Programing Interface
JDBC	Java™ EE Database Connectivity
DFS	Distributed File System
RDBMS	Relational Database Management System
DOC	Document

SUMÁRIO

1 INTRODUÇÃO

No mundo da tecnologia a utilização de dados vem aumentando cada vez mais, a sociedade como um todo vem usando ferramentas que manipulam estes dados gerados e extraem cada vez mais informações inteligentes destes. No site *Data.gov*[1], portal de dados abertos do governo federal dos EUA, o número de arquivos de dados indexados cresceu 400% no ano de 2016.

Dados abertos são dados acessíveis por todos, disponibilizados para obtenção de qualquer pessoa que tenha interesse (Office, 2012), o que não necessariamente significa que são fáceis de utilizar. Os dados abertos geralmente não seguem um padrão, os conjuntos de dados disponibilizados podem ter estruturas diferentes, tipos de valores distintos, salvos e disponibilizados em formatos diferentes como CSV, JSON, SQL, dificultando a análise conjunta de múltiplas bases de dados como MYSQL ao mesmo tempo e/ou combinadas.

Com o crescimento da internet, a quantidade de dados gerados e disponibilizados tem aumentado muito, esses dados estão em sua maioria em formato não estruturado - o que limita sua descrição e reutilização por pessoas e outras aplicações.

Pensando em dados abertos governamentais, o impacto pode ser ainda maior visto que as análises podem gerar valor para toda a sociedade e melhorar a vida de muitas pessoas. No caso de diversos conjuntos de dados distintos é necessário olhar as relações entre estes dados.

Um problema comum que existe é encontrado na utilização de conjunto de dados com formatos e esquemas heterogêneos, essa utilização inclui desde a leitura, verificação e até o tratamento das bases. Como os dados abertos podem estar disponibilizado em diverso e também podem estar disponibilizado em sites distintos, ao acessar um arquivo de texto como base dados, isso se torna uma operação bastante complexa. Além da complexidade para acessar os dados é possível que exista algumas discrepâncias e inconsistências entre as bases. Com isso foi proposto uma arquitetura que trouxesse uma solução para estes problemas com a exceção da extração das bases.

1.1 MOTIVAÇÃO E OBJETIVO

Tendo em vista os problemas relacionados a utilização de conjuntos de dados com formatos e fontes distintos, buscamos com o auxilio de algumas ferramentas resolver esses problemas.

A abordagem deste trabalho apresenta uma solução usando Apache Drill (Foundation, 2020) como ferramenta, a qual possibilita fazer integrações acessando múltiplas bases de dados abertos e com foco principal em fazer junção entre essas bases. São executadas consultas

[1] site:https://www.data.gov/

SQL usando diferentes tipos de junções em tabelas relacionadas. Um dos desafios do sistema apresentado é a capacidade de acessar as bases de dados relacionadas quando essas estão em formatos e fontes distintas. Criamos uma arquitetura visando possibilitar principalmente a experimentação e teste de soluções existentes de integração de dados.

O objetivo final dessa experimentação é verificar a utilidade de diversos formatos usados na disponibilização de dados abertos.

1.2 ESTRUTURA DO DOCUMENTO

Na próxima seção falaremos da fundamentação teórica, que nos dará uma breve visão sobre dados abertos e bases de dados. Daremos também uma explicação simples sobre o que é integração de dados.

Na seção 3, Trabalhos relacionados apresentamos as abordagens mais importantes de diversos trabalhos relacionados à integração de dados.

Na seção 4, Solução de integração de dados descreveremos quais os recursos usados para resolver o problema da integração de dados, (Apache Drill, API) e exemplificamos com a cláusula SQL o que foi utilizado nos testes de junções de tabelas.

2 FUNDAMENTAÇÃO TEÓRICA

Os conceitos explorados pelos trabalhos relacionados e que são utilizados na solução proposta no presente trabalho são abordados nessa seção.

2.1 BANCO DE DADOS

Banco de dados é definido como uma coleção de dados relacionados e de programas que permitem aos usuários acessar e modificar esses dados (SILBERSCHATZ, 2020). Um Sistema de Gerenciamento de Banco de Dados (SGBD) é geralmente quem controla um banco de dados. Os dados, o SGBD e aplicativos associados juntamente são chamados de sistema de banco de dados (Gehrke, 2007). O papel principal de um sistema de banco de dados é fornecer aos usuários uma visão abstrata dos dados, ou seja, o sistema mostra apenas detalhes de como os dados são armazenados e mantidos (SILBERSCHATZ, 2020).

Um SGBD tem como principais características facilitar o controle de bancos de dados, monitorar o desempenho, e realizar tarefas como ajustes, backup e recuperação (Gehrke, 2007). Um banco de dados pode compreender vários modelos, e que o relacional é apenas um deles que é o mais usado. O modelo relacional é muito simples e elegante: cada relação é uma tabela com linhas e colunas (Gehrke, 2007). Como exemplo de um estrutura relacional veja a Tabela 2.1. Essa tabela mostra registros básicos de pessoas com nome, idade e profissão. Há também um identificador único (Id) para permitir a diferenciação dessas pessoas.

Tabela 2.1: Um exemplo de uma tabela de dados do modelo relacional.

Id	Nome	Idade	Profissão
1	Luiz	1	Padeiro
2	Bibinson	27	Músico
3	João	23	Vendedor
4	Robson	44	Professor

Diversas são as relações entre tabelas que podem existir em um banco de dados. São elas: do tipo um para um, um para muitos e muitos para muitos (FAIS, 2009). Para organizar as relações dessas tabelas precisamos de chaves que irão representar cada tabela. Existem dois tipos de chaves, chave primária é um chave única que identificar cada linha de uma tabela, já a chave estrangeira é atribuído a uma linha ou varias linhas de uma tabela, que é chave primária em outra tabela (DAMAS, 2007) , as relações são possíveis além destas chaves (Cury, 2015).

O relacionamento **um para um** se dá de forma direta entre duas tabelas (FAIS, 2009) e acontece quando uma chave primária da linha de uma determinada tabela pode ser utilizada uma única vez em uma outra tabela. No exemplo da Figura 2.1, temos duas tabelas: uma para cadastro de alunos e outra para os documentos (cpf, rg) desse alunos. Neste caso, o código

de cada aluno poderá ser especificado uma única vez na tabela de documento, sendo que cada documento existirá apenas para um aluno (Pichetti, 2021).

Aluno		
Id	**Nome**	**Idade**
1	João Galber	23
2	Pedro da Silva	19

Documentos		
RG	CPF	**Aluno_Id**
102514	123456789-22	2
102513	123452222-23	1

Figura 2.1: Exemplo de uma relação um para um.

O relacionamento **um para muitos** se dá de forma direta entre duas tabelas sempre que a chave primária da linha de uma determinada tabela é utilizada várias vezes em outra tabela, (FAIS, 2009). O próximo exemplo da Figura 2.2 mostra a relação entre uma tabela para alunos e uma tabela para telefones. Neste caso, um mesmo aluno pode ter vários telefones, podendo o seu código ser informado várias vezes em diferentes linhas da tabela de telefone (Pichetti, 2021).

Aluno		
Id	**Nome**	**Idade**
1	João Galber	23
2	Pedro da Silva	19

Telefone		
Id	**Numero**	**Aluno_Id**
1	4399831867	1
2	4399785674	1

Figura 2.2: Exemplo de uma relação um para muitos.

O relacionamento **muitos para muitos** se dá de forma indireta entre duas tabelas, pois é necessário a geração de uma terceira tabela (Setzer, 2005). Ocorre sempre que surge a necessidade de se relacionar duas chaves primárias de linhas de diferentes tabelas em várias linhas de uma terceira tabela (FAIS, 2009). O exemplo mostrado na Figura 2.3 considera que um aluno de uma universidade possa ter muitos professores. Neste caso, tem-se uma tabela para cadastro de professores, uma tabela para cadastro de alunos e uma tabela para alunos_professores. Neste caso na tabela alunos_professores, um mesmo aluno pode ser relacionando com muitos professores e um mesmo professor pode ser relacionado com muitos alunos. Com isso, surgem duas relações um para muitos, que ganha o sentido de muitos para muitos.

Alunos_professores	
Aluno_Id	**Professor_Id**
1	1
1	2
2	1
2	2

Aluno		
Id	**Nome**	**Idade**
1	João Galber	23
2	Pedro da Silva	19

Professor		
Id	**Nome**	**Idade**
1	André Luiz	40
2	Lucas Da Silva	43

Figura 2.3: Exemplo de uma relação muitos para muitos.

É importante conhecer bem as bases antes de fazer qualquer tipo de consulta, isso permite buscar exatamente o que precisamos. Um exemplo disso, é quando fazemos um JOIN e necessitamos utilizar as chaves para comparar atributos e assim fazer a junção desejada.

2.1.1 Join Entre Tabelas

A operação de junção permite unir duas tabelas-argumento que são as tabelas de entrada. É feito o cruzamentos dos dados dessas tabelas, sendo que uma está localizada a esquerda e a outra a direita. A tabela-resultado que é a saída da consulta usando JOIN recebe todas as colunas das duas tabelas-argumento e as linhas dessa tabela são aquelas que satisfazem uma condição desejada (Setzer, 2005). A condição desejada é uma comparação de um determinado valor de uma tabela com um valor da outra tabela. Desta forma as linhas de uma tabela são concatenadas a algumas linhas da outra tabela. Ao escolher adequadamente essas condições, pode-se especificar acessos ao Banco de Dados que envolvem ligações entre tabelas .

Tabela 2.2: Duas tabelas-argumento, tabela Aluno à esquerda e tabela Disciplina à direita.

DisciplinaID	NomeAluno
1	Pablo
3	John
3	Pedro
4	João
4	Daniel
NULL	Samuel

DisciplinaID	NomeDisciplina
1	Física
3	Engenharia
4	Informática
5	Matemática

2.1.2 INNER JOIN

A operação INNER JOIN usa duas tabela-argumento ilustradas na Tabela 2.2, comparando todas as linhas de uma tabela com todas as linhas da outra tabela, buscando encontrar todos os pares de linhas em que a condição de junção é satisfeita (DAMAS, 2007). Caso a condição seja avaliada como verdadeira, ambas as linhas que satisfizeram a condição são concatenadas em um nova linha, que é incluída no conjunto de resultados.

```
SELECT aluno.NomeAluno, aluno.DisciplinaID, disciplina.NomeDisciplina
FROM aluno INNER JOIN disciplina
ON aluno.DisciplinaID = disciplina.DisciplinaID;
```

Tabela 2.3: A consulta sql usando a cláusula INNER JOIN.

DisciplinaID	NomeAluno	DisciplinaID	NomeDisciplina
1	Pablo	1	Física
3	John	3	Engenharia
3	Pedro	3	Engenharia
4	João	4	Informática
4	Daniel	4	Informática

2.1.3 LEFT JOIN

A operação LEFT JOIN usa duas tabelas-argumento ilustradas na (Tabela 2.2) (que sera chamada de tabela esquerda e tabela direita). Essa operação compara todas as linhas da tabela esquerda com todas as linhas da tabela a direita. Caso essa comparação seja verdadeira a linha da tabela esquerda será concatenado com a linha da tabela direita para formar uma nova linha no conjunto de resultados (Pichetti, 2021). Caso contrário a consulta irá manter os valores das colunas da tabela esquerda e preencherá com NULL para cada coluna da tabela direita.

```
SELECT aluno.NomeAluno, aluno.DepartamentoID, disciplina.NomeDisciplina
FROM aluno LEFT JOIN disciplina
ON aluno.DepartamentoID = disciplina.DepartamentoID;
```

Tabela 2.4: A consulta sql usando a cláusula LEFT JOIN.

DisciplinaID	NomeAluno	DisciplinaID	NomeDisciplina
1	Pablo	1	Física
3	John	3	Engenharia
3	Pedro	3	Engenharia
4	João	4	Informática
4	Daniel	4	Informática
NULL	Samuel	NULL	NULL

2.1.4 RIGHT JOIN

A operação RIGHT JOIN usa duas tabelas-argumento ilustradas na (Tabela 2.2) (que sera chamada de tabela esquerda e tabela direita). Essa operação compara todas as linhas da tabela esquerda com todas as linhas da tabela a direita. Caso essa comparação seja verdadeira a linha da tabela direita será concatenado com a linha da tabela esquerda para formar uma nova linha no conjunto de resultados (Pichetti, 2021). Caso contrário a consulta irá manter os valores das colunas da tabela direita e preencherá com NULL para cada coluna da tabela esquerda.

```
SELECT aluno.NomeAluno, aluno.DepartamentoID, disciplina.NomeDisciplina
FROM aluno RIGHT JOIN disciplina
ON aluno.DepartamentoID = disciplina.DepartamentoID;
```

Tabela 2.5: A consulta SQL usando a cláusula RIGHT JOIN.

DisciplinaID	NomeAluno	DisciplinaID	NomeDisciplina
1	Pablo	1	Física
3	John	3	Engenharia
3	Pedro	3	Engenharia
4	João	4	Informática
4	Daniel	4	Informática
NULL	NULL	5	Matemática

2.1.5 FULL JOIN

A operação FULL JOIN é basicamente a união da operação LEFT JOIN visto na subseção 2.1.4 e a operação RIGHT JOIN visto na subseção 2.1.3 usa duas tabelas-argumento ilustradas na (Tabela 2.2) (que sera chamada de tabela esquerda e tabela direita). A operação FULL JOIN mantém todas as linhas das duas tabelas (Tabela 2.2), correspondidas ou não. FULL JOIN é um tipo de junção externa, mas também pode ser classificado como junção externa completa (Setzer, 2005). Se não existe nenhuma correspondência para qualquer linha da tabela esquerda, então as colunas da tabela direita serão nulas. Isso também acontecerá caso não existir correspondência para as linhas da tabela direita.

```
SELECT aluno.NomeAluno, aluno.DepartamentoID, disciplina.NomeDisciplina
FROM aluno FULL JOIN disciplina
ON aluno.DepartamentoID = disciplina.DepartamentoID;
```

Tabela 2.6: A consulta sql usando a cláusula FULL JOIN.

DisciplinaID	NomeAluno	DisciplinaID	NomeDisciplina
1	Pablo	1	Física
3	John	3	Engenharia
3	Pedro	3	Engenharia
4	João	4	Informática
4	Daniel	4	Informática
NULL	NULL	5	Matemática
NULL	Samuel	NULL	NULL

2.1.6 CROSS JOIN

A operação CROSS JOIN usa duas tabelas-argumento ilustradas na (Tabela 2.2) (que sera chamada de tabela esquerda e tabela direita), o retorno é um produto cartesiano dessas duas tabelas, ou seja, todas as linhas de uma tabela com a outra vão ser associadas (DAMAS, 2007). Se adicionar uma cláusula WHERE na consulta, com isso o resultado ficará igual ao do INNER JOIN. Neste caso, não existe motivo para usar CROSS JOIN se o resultado é similar ao INNER JOIN.

```
SELECT aluno.NomeAluno, aluno.DepartamentoID, disciplina.NomeDisciplina
FROM aluno CROSS JOIN disciplina
WHERE aluno.DepartamentoID = disciplina.DepartamentoID;
```

Tabela 2.7: A consulta sql usando a cláusula CROSS JOIN.

DisciplinaID	NomeAluno	DisciplinaID	NomeDisciplina
1	Pablo	1	Física
3	John	3	Engenharia
3	Pedro	3	Engenharia
4	João	4	Informática
4	Daniel	4	Informática

2.2 DADOS ABERTOS

Segundo (Office, 2012), dados abertos são dados que: são acessíveis sem limitações relacionadas à permissões para usuários, o único custo existente é custo próprio de manipulação

destes dados; estão disponibilizados digitalmente e em um formato que é possível que a leitura seja feita por uma máquina para que os mesmos possam ser integrados com outros dados de fontes diferentes; são livre de restrições no uso e redistribuição em seu termo de uso. Uma das habilidades mais desejadas nos últimos anos é a de saber trabalhar com grandes quantidades de dados provenientes de diversos locais e em diferentes formatos (Seiji Isotani, 2015).

O consumo de dados abertos pode ser feito por públicos de diferentes perfis, desenvolvedores tendem a utilizar APIs, pois elas facilitam o consumo automatizado sobre os dados, já outras pessoas tendem a preferir formatos mais comuns como CSV, DOC, que são mais fáceis de compreender (Williams Alcantara, 2015).

Quando falamos de Dados Abertos o tipo de arquivo CSV e JSON estão entre os mais utilizados para publicação de conjuntos de dados. Temos como exemplo os conjuntos de dados disponibilizados pelo governo brasileiro em (DADOS-ABERTOS, 2021). Analisamos também o portal de dados abertos do Kaggle (Kaggle, 2021). Com isso, conseguimos ver um grande uso destes formatos para disponibilização de dados abertos. Nas próximas subseções vamos abordar os principais formatos de arquivos utilizados no presente trabalho.

2.2.1 Dados em Formato CSV

CSV (Comma-Separated Values) significa valor separado por vírgula. Cada linha de um arquivo CSV é um registro de dados. Esses registro são composto de um ou mais campos separados por vírgulas. O arquivo CSV não é completamente padronizado. Originalmente era utilizada apenas a vírgula para separar valores e a quebra de linhas para separação dos registros (Shafranovich, 2005).

Observando a Tabela 2.8, temos um exemplo de arquivo csv em formato tabela. Já na Figura 2.4, temos um exemplo de arquivo csv em formato texto. As linhas do arquivo de texto são delimitadas por quebra de linha, os campos nas linhas são delimitados por vírgulas e cada campo na linha é uma coluna. Portanto na tabela os campos são formatos pela interseção de linhas por colunas, campo é equivalente a célula (Shafranovich, 2005).

Tabela 2.8: Arquivo CSV em formato tabela..

Id	NomeCurso
31	Física
33	Engenharia
34	Informática
35	Matemática

```
Id,NomeCurso
31,Física
33,Engenharia
34,Informática
35,Matemática
```

Figura 2.4: Arquivo CSV em formato texto.

2.2.2 Dados em Formato JSON

Segundo o livro do autor (SILBERSCHATZ, 2020):

> JavaScript Object Notation (JSON) é uma representação textual de tipos de dados complexos bastante utilizada para transmitir dados entre aplicações e para armazenar dados complexos. JSON suporta os tipos de dados primitivos integer, real, string, bem como arrays e "objetos", que são uma coleção de pares do tipo (nome do atributo, valor).

Um JSON é composto de um ou mais elementos onde cada elemento é formado por um ou mais atributos. Os nomes dos atributos são chaves e são sucedidos por dois pontos (:), e seus valores associados ficam à direita. Como exemplo, na Figura abaixo estão os dados de um arquivo em formato JSON. O conteúdo deste arquivo consiste de um vetor de elementos (basicamente uma lista, representada em colchetes). Esse vetor contém 3 elementos (cada um representado por chaves), onde cada um possui 3 atributos: id (tipo inteiro), nome (tipo texto) e idade (tipo inteiro). As informações sobre os dados do arquivo é representando da seguinte forma: id 1 representa o Luiz Cavalheiro que tem 13 anos, id 2 representa o Fernando S que tem 11 anos e o id 3 representa o Bibinson Celissaint que tem 15 anos).

```
[{
   "id":1,
   "nome":"Luiz Eduardo",
   "idade":13
},
{
   "id":2,
   "nome":"Fernando S",
   "idade":11
},
{
   "id":3,
   "nome":"Bibinson Celissaint",
   "idade":15
}]
```

2.3 INTEGRAÇÃO DE DADOS

A integração de dados é o problema de relacionar dados que estão em fontes distintas e fornecer ao usuário uma visão unificada desses dados. Isso traz um desafio complexo para as organizações que implantam arquiteturas de Big Data (Hariharan, 2014).

Um dos tipos de integração de dados é processo ETL (Extract, Transform, Load) que é divido em: etapa de extração que executa a recuperação de dados das fontes, etapa de carregamento que aplica os recursos necessários antes de carregar os dados no banco de dados e garante o uso mínimo dos recursos, e por fim a etapa de transformação que envolve a implementação de um conjunto de regras para transformar os dados da origem para o destino (Atiq, 2015).

Em sistemas de integração baseados em SQL, é comum na eliminação de registros duplicados agrupar por operador usando os atributos-chave dos registros em combinação com funções agregadas para reconciliar valores de atributos não-chave divergentes (Eike Schallehn, 2003). As primeiras soluções na área de integração de bancos de dados relacionais, visavam este conceito que foi expandido para diferentes fontes de dados, como arquivos em formatos: CSV, XML e texto (del Fabro, 2018).

Uma arquitetura comum para integração de dados é a mostrada pela Figura 2.5, na qual temos três fontes de dados diferentes (CSV, JSON, MYSQL), que são acessados por um ponto de acesso no caso que é o Apache Drill. O usuário tem uma ligação com o ponto de acesso que o permite ter acesso aos três fontes de dados, portanto, consegue realizar uma junção entre dados requisitados por consultas sobre os dados reais nas fontes.

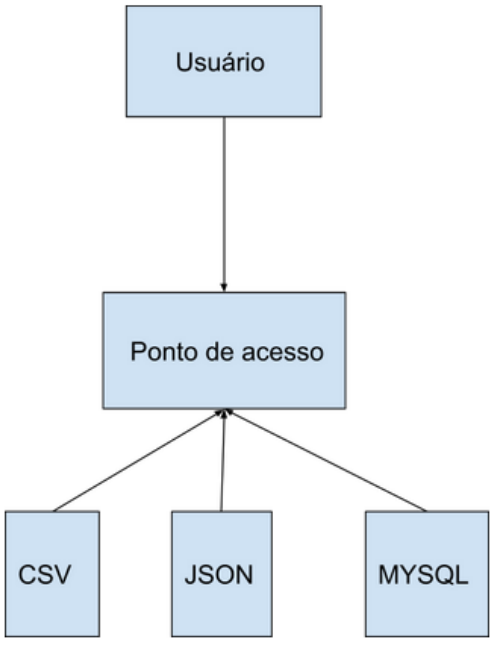

Figura 2.5: Arquitetura das bases de dados com fontes diferentes.

2.4 TECNOLOGIAS EXISTENTES

Existem muitas tecnologias para integração de dados, no entanto iremos abordar apenas as que utilizamos em nosso trabalho, são elas: MYSQL, SQL, API e Apache Drill. Nesta seção vamos definir cada um dos conceitos citados.

O MySQL é um sistema gerenciador de banco de dados relacional de código aberto. Ele na maioria das vezes é utilizado para gerenciar bases de dados nas aplicações gratuitas. A linguagem mais utilizada para acessar, inserir e gerenciar as informações de um banco de dados é o SQL (Structure Query Language) (Pisa, 2012). SQL é uma linguagem de programação usada por diversos bancos de dados relacionais (Oracle, 2014).

A linguagem SQL é formada por diversas cláusulas, onde cada cláusula é definida como uma palavra em uma frase. A combinação das cláusulas se torna uma instrução SQL, na qual cada uma tem sua função. Em uma instrução SQL comum é possível encontrar diversas cláusulas como: SELECT, FROM, WHERE, ORDER BY, entre outras. Dessas cláusulas apenas o SELECT e o FROM são obrigatórios (Microsoft, 2021).

É possível utilizar um SQL em uma API para acessar as informações em uma ou mais base de dados. A sigla API (Application Programming Interface) é um conjunto de regras que permite a comunicação entre plataformas através de uma série de padrões e protocolos (Fabro, 2020). Por meio de uma API é possível criar novos aplicativos e softwares capazes de se comunicar com outras plataformas (Fabro, 2020). Pode ser considerado então uma API como: uma ponte para acessar determinadas informações armazenadas em uma base de dados. Uma das ferramentas que utiliza API para troca de dados é o Apache Drill.

Apache Drill fornece uma API que recebe uma requisição que contém um SLQ que permite obter o resultado sobre a base acessada pelo Apache Drill. O Apache Drill é um estrutura de software de código aberto que suporta aplicativos distribuídos. O Drill foi projetado para oferecer suporte a análises de alto desempenho nos dados semiestruturados. Ele fornece integração plug-and-play com as implantações existentes do Apache Hive e do Apache HBase (Foundation, 2020).

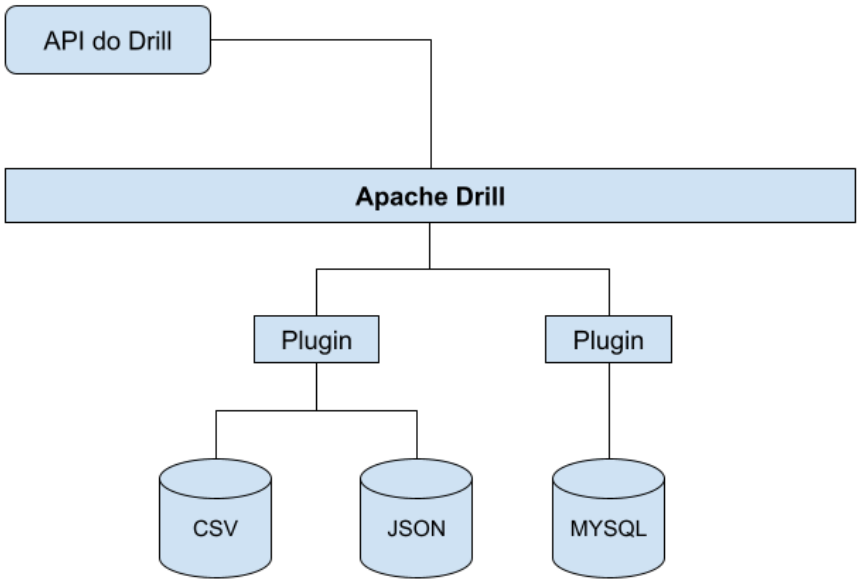

Figura 2.6: Arquitetura do Apache Drill.

A figura acima é uma ilustração da arquitetura do Apache Drill. A API do Apache Drill nos permite mandar requisições ao sistema do Drill como por exemplo um objeto em formato JSON, que é composto de dois elemento onde um é chamado "queryType"o qual recebe o tipo da consulta e o outro que é chamado "query"e recebe o SQL da consulta. No Apache Drill existem plugins que nos permitem ter acesso a um subconjunto de bases como MYSQL ou as tabelas de formato CSV e JSON, a maior parte do plugins é fornecido pelo Apache Drill que é preciso apenas configurar. Como exemplo temos o plugin que faz acesso ao MYSQL é necessário configurá-lo com as próprias informações da base MYSQL escolhida. No caso do plugin que faz o acesso para o CSV e JSON é fornecido a configuração pelo Apache Drill.

2.5 CONCLUSÃO

Nesta seção definimos vários conceitos utilizados no trabalho como Banco de Dados, Dados Abertos, Integração de Dados e Apache Drill. Na próxima seção será mostrado um subconjunto de artigos utilizado como base para o trabalho e uma tabela comparativa com as principais características de cada artigo.

3 TRABALHOS RELACIONADOS

Nesta seção apresentamos uma visão geral sobre integração de dados. Na subseção 3.1, apresentamos uma tabela comparativa das principais caraterísticas de cada artigo e que estão relacionadas com o nosso trabalho e na subseção 3.4 concluímos.

De acordo com este artigo (Hariharan, 2014) a quantidade enorme de dados criados e mantidos por indústrias, instituições de pesquisa estão à beira de expandir sua infraestrutura. Os avanços no fluxo de trabalho da organização incluem armazenamento, gerenciamento, manutenção, integração e interoperabilidade de dados. Entre esses níveis, integração e interoperabilidade de dados são as duas principais áreas de foco para as organizações que tendem a implementar avanços em seus fluxos de trabalho. No geral, essas áreas influenciam o desempenho da organização, sendo desafios complexos para as organizações que implantam arquiteturas de Big Data devido a heterogeneidade dos dados usados por eles. Portanto, requer uma abordagem abrangente para negociar os desafios na integração e interoperabilidade.

Segundo o artigo (Atiq, 2014), muitos tipos de integração de modelo precisam ser analisados com eficiência em Big Data o que requer um modelo extremamente preciso. A integração de dados é o problema de combinar dados que podem estar localizados em diversos fontes e fornecer ao usuário uma visão unificada desses dados. A integração de modelos existentes extrai o mapeamento de modelos de baixa qualidade em grandes bancos de dados, e os sistemas se concentram apenas na identificação de modelos úteis no nível do valor do atributo. Eles fornecem uma técnica generalizada para permitir a integração correta de múltiplas fontes de dados. Melhora consideravelmente a qualidade da reorganização do modelo.

Em (Miller, 2018) um ponto principal que foi abordado é sobre dados abertos, que são dados estruturados legíveis por máquina, usados e construídos sem restrição. O ponto mais importante no caso dos dados abertos, é que muitos governos federais, provinciais e municipais estão adotando o dados abertos. Organizações privadas e públicas estão usando esses princípios quando transparência é de interesse nos negócios. Baseado na observação do (Renée J. Miller), pelo menos 28 países ao redor do mundo estão publicando quantidades substanciais de dados abertos. No data.gov, portal de dados abertos do governo federal dos EUA, o número de arquivos de dados indexados cresceu 400% no ano anterior a março de 2017.

Em (Atiq, 2015) o autor usa ETL (Extract Load Transform), que é um processo onde os dados são extraídos, carregados e transformados em um conjunto de dados gigante no final do processo de integração. O tamanho do conjunto pode afetar o capacidade de armazenamento da base. Esse processo é recomendado para integração de dados, pois garante uma abordagem sistemática e incremental de integração.

Em (Lenzerini, 2002) os autores tratam o problema sobre uma estrutura lógica para integração de dados. Para eles a integração de dados tem como objetivo combinar os dados

residentes em diferentes fontes e fornecer ao usuário uma visão unificada, visão essa que representa um esquema global e auxilia o usuário em suas consultas sobre esses dados.

Em (Eike Schallehn, 2003) os autores tratam de um problema que pode acontecer com o uso de operações SQL como agrupamento (GROUP BY) e junção (JOIN). Estas operações falham se os valores de atributo das duplicatas potenciais ou tuplas relacionadas não forem iguais, mas apenas semelhantes por certos critérios. Como uma solução para esse problema, eles propõem uma variante baseada em similaridade de operadores de agrupamento e junção.

3.1 TABELA COMPARATIVA

Existem vários trabalhos e artigos relacionados a integração de dados, neste trabalho focamos em um subconjunto restrito de trabalhos que tratam de integração de dados abertos, onde os dados são integrados usando métodos de similaridade, apresentamos então uma tabela comparativa sobre um subconjunto trabalhos com intuito de mostrar as principais diferenças entre eles.

Tabela 3.1: Tabela comparativa entre os trabalhos relacionados.

	Fonte dos dados abertos	Quantidade de fontes	Método Join	Inconsistencia dos dados	Processo ETL
Similarity-based operations (Eike Schallehn, 2003)	Base de dados	Múltiplos	Eliminação de duplicatas	Discrepâncias	Limpeza de dado
Progression in Large Fonte de Dados (Atiq, 2014)	Múltiplas bases na Web	Múltiplos	Combinação de dados	Restrições de integridade	Transformação de dados
Challenges of Data Integration (Atiq, 2015)	FlatFile	Múltiplos	Não se aplica	Recursos inadequados	Extração de dados
Open Data Integration (Miller, 2018)	CSV, JSON	Múltiplos	Busca de união em tabelas	Não se aplica	Transformação de valores para a junção
Data Integration: A Theoretical Perspective (Lenzerini, 2002)	XML	Múltiplos	Combinação de dados	Restrições de integridade	Transformação adequada, Procedimentos de limpeza

Mesmo que os trabalhos apresentarem abordagens bastante diferentes, existe um ponto em comum na integração de dados que é Múltiplas fontes de dados. No (Eike Schallehn, 2003) os autores falam sobre o problema de operação baseado na similaridade em múltiplas fontes de dados, usando base de dados abertos. Algumas bases podem gerar duplicação de dados, para usar o método da junção é necessário aplicar a eliminação de duplicação, que é considerado como uma limpeza dos dados. Algumas das abordagem tratadas em Progression in Large Fonte de Dados (Atiq, 2014) é a transformação de dados em grande base de dados aberto, combinando dados para facilitar a junção entre múltiplos fonte de dados na web. E os autores tratam o problema da inconsistência de dado como restrições de integridade. A extração de dados é uma abordagem muito importante, foi usado no Challenges of Data Integration (Atiq, 2015) para permitir o acesso aos diferentes tipo de dados como Flatfile que é um tipo dado em formato de arquivo, SQL, XML e mais algum outro tipo de base de dados. Recurso inadequado é uma das abordagens tratada em (Atiq, 2015), que é um problema que pode ocorrer ao relacionar os dados. Em Open Data Integration (Miller, 2018), é preciso fazer a transformação dos valores para a junção, em algumas bases os valores aparecem em tipo alfa-numérico, já em outras bases são do tipo numéricos, buscando então a união de tabela em múltiplos fontes de dados como arquivos CSV e arquivos JSON. No Data Integration: A Theoretical Perspective (Lenzerini, 2002) é

feito uma transformação adequada e um procedimentos de limpeza para deixar os dados mais acessíveis e permitir assim aplicar outros tipo de operações como por exemplo a combinação de dados em múltiplos fontes de dados tal como XML.

Os trabalhos apresentam abordagens que ajudam a fazer a integração dos dados, tal como acessar as bases de dados em diferente fontes de dados, porém, não existe um protótipo usado como solução. Na maioria dos trabalhos os autores falam sobre a implementação de algum método em específico como JOIN, GROUP BY e UNION entre tabela, mas não como acessar os dados.

É possível observar várias diferenças entre os trabalhos relacionados, um dos pontos em comum é que todos trabalham com fonte de dados múltiplos e isso é um dos motivos para fazermos a integração do dados. Com várias fontes de dados é normal que apareçam problemas, com a integração de dados ao combinar dados, conseguimos eliminar a duplicação por exemplo. Alguns trabalhos usam o próprio banco de dados para fazer a integração e outros usam Flatfile como CSV, JSON e XML. O tratamento dos dados é uma etapa bem importante, tal como o ETL que permite carregar extrair e transformar os dados.

3.2 CONCLUSÃO

Todos os trabalhos lidos tratam de assuntos diferente dentro do universo de integração de dados, mas é possível perceber alguns pontos em comum como dados aberto e múltiplos fontes de dados. Cada um dos trabalhos apresentado nesta seção sugeriu uma solução para nosso trabalho, tal como ETL (Extract Load Transform) que é o processo de transformação do dados nas bases e também o conceito de dados abertos que é o tema principal e nos permite acessar as informações disponibilizado pelas instituições. A utilização do SQL é bem importante no nosso trabalho, pois nos ajuda a executar uma consulta de forma correta usando a cláusula JOIN. O problema da integração é complexo, neste trabalho será estudada uma solução existente que é o Apache Drill, a qual será utilizada para integração de dados abertos em diferentes formatos.

4 INTEGRAÇÃO DE DADOS ABERTOS USANDO APACHE DRILL

Nessa seção falaremos sobre a arquitetura utilizada em nosso trabalho e a implementação criada, tendo em vista que é um trabalho prático para experimentar soluções existentes de integração de dados com formatos e esquemas heterogêneos.

4.1 ARQUITETURA

Tendo a necessidade de possuir um sistema modularizado e abstrato o suficiente para que seja acessível e utilizável em diversos cenários com diferentes conjuntos de dados, uma arquitetura clara e com módulos bem definidos é essencial. A arquitetura da solução proposta é classificada em componentes internos e externos, sendo os internos os componentes que fazem parte do sistema junção de dados e os externos os componentes de entrada e de saída. Na sequência, os componentes internos e externos serão abordados detalhadamente.

Como componentes externos, destacamos os arquivos CSV, JSON e as tabelas de um banco de dados relacional que são usados como entradas e as possíveis junções que são retornadas como saída. Como componentes internos, podemos destacar os módulos de leitura de arquivos CSV, JSON e as de tabelas de bancos de dados relacionais. Existe também os módulos de verificação das colunas, tratamento das colunas e o de correlação entre colunas. Podemos visualizar esta arquitetura e o fluxo de informações na Figura 4.1.

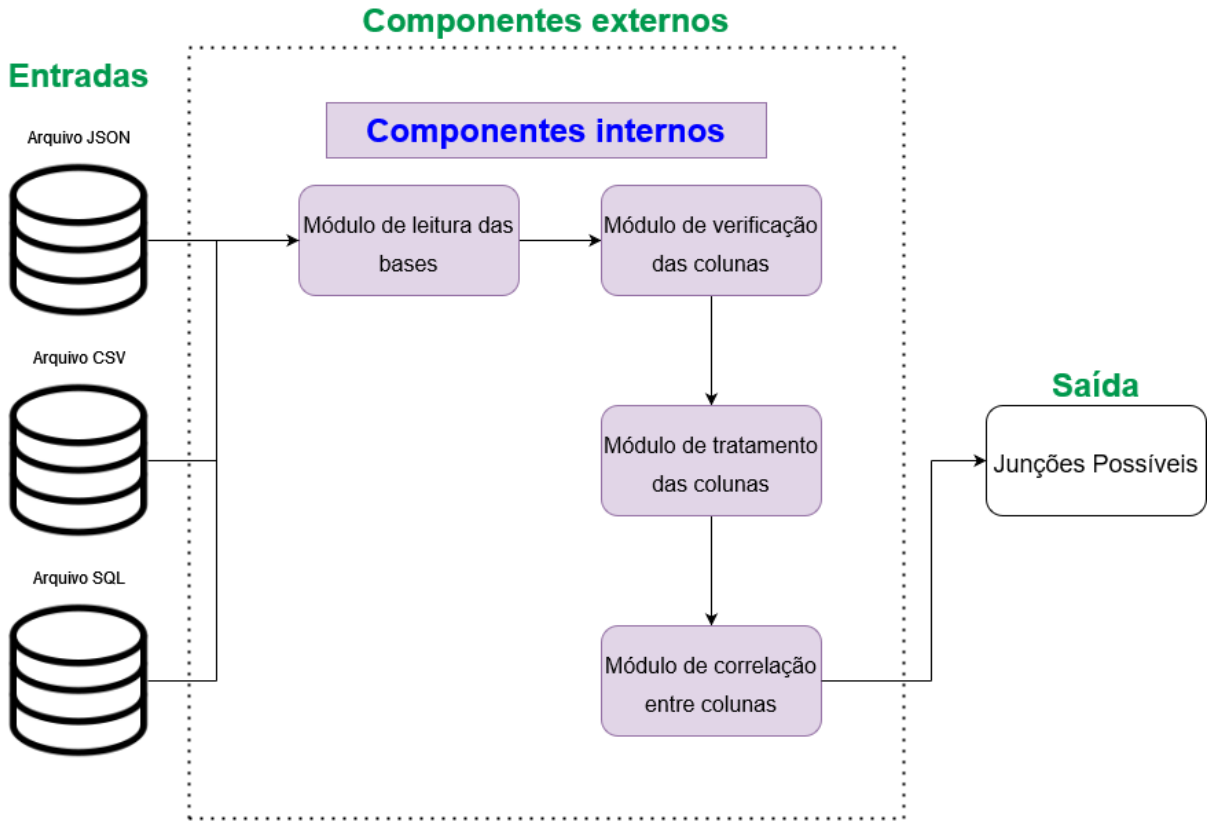

Figura 4.1: Arquitetura e fluxo de dados da solução.

4.1.1 Componentes externos

Como entradas, temos a possibilidade de leitura de arquivos CSV, JSON e SQL. Dessa forma, podemos ter como entrada qualquer base que esteja nesses formatos e as instâncias das bases são a aplicação desta arquitetura. Além de escolher esses três tipos de bases, poderíamos escolher outras bases, porém para simplificar o nosso experimento usamos somente estes formatos que são os arquivos CSV, JSON e SQL. Após a leitura destas base elas serão processadas e gerarão uma saída conforme a arquitetura do trabalho, ou seja uma junção entres as bases lidas.

4.1.2 Componentes internos

Os componentes internos são divididos em: módulo de leitura, módulo de verificação das colunas, módulo de tratamento das colunas e módulo de correlação entre bases.

O módulo de leitura das bases é responsável pela leitura dos arquivos JSON, CSV e SQL. Esse módulo é extremamente importante porque é nele que é possível entender como os arquivos estão estruturados. Esses arquivos são lidos individualmente, sendo que cada tipo de arquivo tem suas características e precisa ser analisado. No caso do CSV é mais complexo, pois nele existem muitas discrepâncias como por exemplo tipo dos valores, nome das colunas, caso precisa usa uma coluna de um arquivo CSV é necessário descobrir qual é posição desta coluna, pois ao fazer uma consultar é retornado uma array que contem o nome de todas as colunas da

base. Em bases como JSON e SQL ao executar um consulta geral como **select from * table**, todos os dados são retornados. Nesta fase é importante saber onde se encontra cada base e seus respectivos tipos.

O módulo de verificação das colunas é responsável por fazer a análise das bases e tabelas e obter informação relevante sobre os dados, é importante analisar todas as colunas das tabelas para descobrir qual é o tipo e nome delas, também é necessário saber qual das colunas que é chave primária ou chave estrangeira, além disso precisamos saber qual é o relacionamento entre uma tabela e outra. Como as bases não são de fonte comum, é possível existir uma dificuldade maior para fazer a correlação entre as colunas das tabelas, pois temos que obter as colunas que tem relação entre si. A verificação das base deve ser feita para cada uma em particular, pois é nesta fase que é possível descobrir qualquer tipo de tratamento necessário. Esta etapa é importante porque os usuários do Dados Abertos não sabem como os dados estão armazenados neste sentido sempre vai ser necessário usar este módulo. No caso do arquivo CSV é necessário uma verificação mais aprofundada, pois é uma base mais complexo, mais difícil de usar. Assim que termina a parte da verificação, deve seguir para o próximo etapa que é o módulo de tratamento.

O módulo de tratamento das colunas é responsável por tratar os problemas que podem aparecer durante uma consulta. Como sabemos, as base são de fonte diferentes e por isso é possível existir inconsistência entre as correlações no sentido que dois atributos a serem relacionados podem ser de tipos diferentes e neste caso precisamos fazer a conversão sobre esses dados para alinhá-los. Após a verificação é possível saber qual dos atributo são do tipo numérico ou do tipo texto, caso precise de alguma conversão é usado uma função chamada CAST que nos permitir converte um valor de tipo texto em tipo numérico ou numérico em texto. Ao fim desse módulo é possível fazer uma consulta com o uso dos valores sem que ela falhe, como poderia acontecer se fosse comparado um valor do tipo numérico com um valor do tipo texto. Outro ponto importante é usar convenções para facilitar a visualização de cada coluna no arquivo CSV.

O módulo de correlação entre bases é ultimo módulo dos componentes internos. Esse módulo é executado somente depois da execução dos módulos anteriores, nele é possível fazer a junção entre as tabelas relacionadas e executar uma consulta usando JOIN. Para fazer a junção entre as tabelas é importante saber qual é a chave que relaciona as tabelas, isso é função do módulo de verificação, caso seja necessário algum tipo de conversão utilizamos o módulo de tratamento. Com isso fazemos a junção entres as tabelas escolhidas para assim gerar uma saída com as junções possíveis.

4.2 IMPLEMENTAÇÃO

Para desenvolvimento desta solução utilizamos a API do Apache Drill, ela nos fornece uma interface que permite trabalhar com vários plugins. É possível criar um plugin e editá-los através da tela de configuração. É necessário rodar o servidor Apache Drill para que seja possível acessar a página que nos permite configura os plugins, o URL da API recebe as requisições das consultas.

Temos como exemplo o plugin **rdbms**, que é usado para acessar os dados na base **MYSQL**, nele é necessário atualizar os campos que podemos ver na Figura 4.2.

Figura 4.2: Configuração plugin rdbms para acessar base MySQL.

Para acessar uma base do MySLQ em JBDC é necessário fazer uma conexão usando o URL que indica a localização da base, o nome do usuário e senha. É necessário um Drive específico para permitir o acesso ao MySQL. No Apache Drill essas informação também são requeridos como é mostrado na Figura 4.2 para configurar o plugin rdbms.

Em outro plugin chamado de **dfs**, é permitido executar uma consulta para acessar os dados dos arquivos **CSV** e **JSON**, sendo possível ver a configuração do plugin dfs na Figura 4.3.

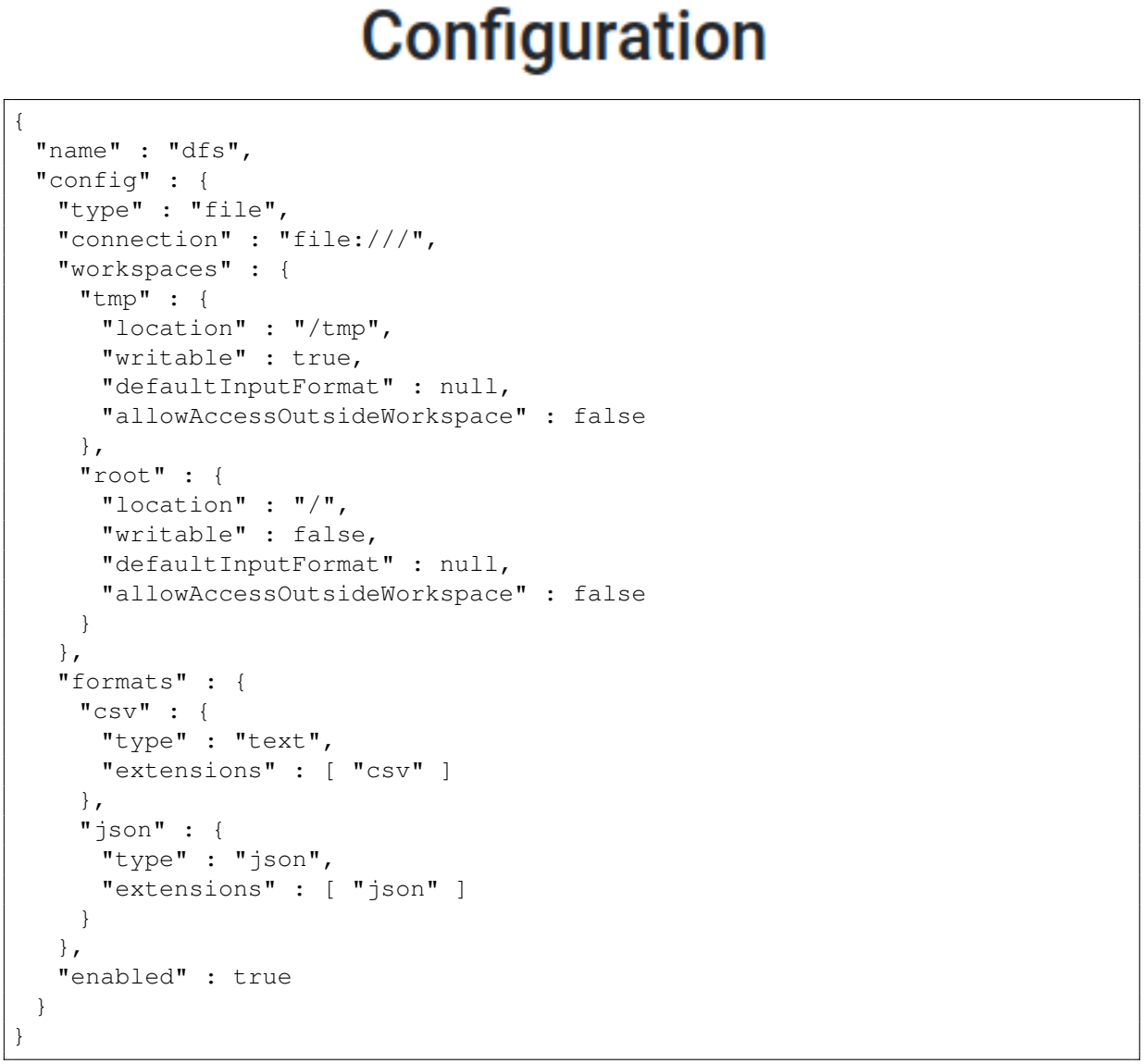

Figura 4.3: Configuração plugin dfs para acessar arquivo em formato JSON e CSV.

A configuração deste plugin é fornecida pelo Apache Drill para diversos formato de arquivos, no caso do nosso trabalho é usado somente como formato CSV e JSON. No exemplo é possível ver na linha 20 que existe um elemento com nome "formats"onde fica todos os tipos de formatos que o o plugin pode acessar e também existe a possibilidade de remover ou adicionar novos formatos. O formato de arquivo é representado como um elemento que contém dois atributos: tipo e extensão.

Cada tipo de plugin é feito para um formato de arquivo ou bases de dados diferentes, com isso temos a possibilidade de habilitar e desabilitar os plugins desejados. Existe um grupo de plugins habilitados que nos permitem executar uma consulta. Caso um plugin esteja desabilitado, não será possível executar uma consulta neste plugin. É importante saber que antes de habilitar a maioria dos plugins é necessário atualizá-los para configura de forma correta o acesso com as bases ou arquivos desejados.

A localização dos dados fica a critério do usuário, podendo estar em qualquer lugar na máquina, basta indicar o caminho completo dos dados.

Para fazer uma consulta é necessário aplicar o padrão do Apache Drill como mostrado no exemplo abaixo.

```
select * from dfs.`/path/dados.json`;
ou
select * from dfs.`/path/dados.csv`;

select * from rdbms.baseMysql.tabelas;
```

Aqui está a descrição da nossa implementação na linguagem Java.

```
1   public static void Execute(String sql) {
2     try {
3       String url = "http://localhost:8047/query.json";
4       URLConnection connection = new URL(url).openConnection();
5       connection.setDoOutput(true); // Triggers POST.
6       connection.addRequestProperty("User-Name", "mapr");
7       connection.setRequestProperty("Content-Type", "application/json");
8       String data = "{\"queryType\":\"SQL\", \"query\": \""+sql+"\"}";
9       try (OutputStream output = connection.getOutputStream()) {
10        output.write(data.getBytes(StandardCharsets.UTF_8.name()));
11      }
12
13      try (InputStream response = connection.getInputStream()) {
14        String result = IOUtils.toString(response);
15        System.out.println(result);
16      }
17    } catch (Exception e) {
18      System.out.println("Something went wrong. "+ e.getMessage());
19    }
20  }
```

Este programa cria uma conexão e faz uma requisição para a API do Apache Drill, é possível ver na linha 4 que existe **URLConnection** o qual permite criar uma instância chamada de **connection**, nesta instância é aberta uma conexão usando o URL do Apache Drill. Na instância **connection** existem várias propriedades que podem ser vistas na linha 5 *connection.setDoOutput(true);* que é usado para mandar uma requisição POST. Na linha 6 e 7 é fornecido as informações do cabeçalho, "User-Name"e "Content-Type". Na linha 8 a variável **data** recebe um objeto do tipo JSON em formato string, existe uma barra invertida para permitir a concatenação dos atributos. Na linha 9,10 e 11 é feita uma requisição na conexão aberta. Na linha 13 é obtida a resposta e em seguida na linha 14 a resposta é atribuído a uma variável **result** como string. Na linha 16 o resultado é imprimido. O trecho de código entre as linha 3 e 16 é fornecido pelo Apache Drill para facilitar o acesso a dados abertos na linguagem Java. Na linha 8 substituímos o SQL por uma variável chamada sql, é possível passar um SQL genérico na consulta como parâmetro da função.

4.2.1 Consulta usando Join

Para fazer esta consulta utilizamos três bases de dados que são: municípios, estados e CEP. As bases municípios e estados foram retiradas de um site do governo brasileiro (DADOS-ABERTOS, 2021), que disponibiliza 10.373 conjuntos de dados (consultado em: 12 de setembro de 2021). A base CEP foi retirada do portal de dados abertos Kaggle (Kaggle, 2021). Com isso, conseguimos ver um grande uso destes formatos para disponibilização de dados abertos. Como

não poderia acessar direto estas bases para fazer uma consulta, foi necessário fazer a integração de dados para facilitar o acesso. Assim foi possível fazer junção das base de dados ou executar outros tipos de consultas em diferentes fontes de dados.

A linha 1 do Algoritmo abaixo é formado por uma variável chamada **select**, essa variável recebe um string composto por uma cláusula select e as colunas escolhidas para mostrar na consulta. Na linha 2 existe uma variável nomeada **from**, onde é atribuído a cláusula from e a tabela escolhida, notem que existe um dfs que é um plugin do Apache Drill que permite acessar diversas bases, nesse presente trabalho foi utilizado para acessar CSV e JSON. Para utilizar plugin é necessário o caminho e nome do arquivo, como é possível ver exemplo a seguir: dfs.‘/caminho/arquivo.tipo‘. Na linha 3 existe mais uma variável com nome **join** que recebe qualquer tipo de join, tal como: inner join, left join, right join, full join nas tabelas escolhidas. Já na linha 4 tem a variável **on**, nela é passado a cláusula on que permite fazer uma comparação para saber quais linhas das duas tabelas em que a condição é satisfeitas, também é importante aplicar o cast o qual permite converter uma coluna do tipo inteiro em string. Na linha 5 existe a variável **where**, que recebe a cláusula where a qual permite colocar algumas restrições na consulta, como: where cep=? "isso é opcional". Na linha 6 existe a variável **sql**, que recebe a concatenação de todas as variáveis criadas, por fim este sql é enviado para uma função que faz a requisição para o Apache Drill.

Algoritmo 1 Join:

1: $select \leftarrow select + colunas$
2: $from \leftarrow from + dfs.‘/path/tableName‘$
3: $join \leftarrow joinType + dfs.‘/path/tableName‘$
4: $on \leftarrow on + CAST(tableName.codigo\_as\_char) = tableName.codigo$
5: $where \leftarrow where + tableName.column =?$
6: $sql \leftarrow select + from + join + on + where$
// “Depois de montar o SQL é possível fazer a requisição nas $bases$
7: executar $consulta(sql)$ nas bases de *Dados*

4.3 RESULTADOS EXPERIMENTAIS

Nesse seção iremos apresentar o estudo de caso e a saída do nosso experimento. No estudo de caso apresentamos as dificuldades encontradas na busca de bases e quais as bases escolhidas. A saída foi obtida através do experimento feito com as bases selecionadas executando consultas com operação JOIN.

4.3.1 Estudo de Caso

Para validar o sistema e testar sua eficácia, o experimento foi baseado no Algoritmo 1, utilizamos as bases de localização do Brasil que foi citado nas seções anteriores. Escolhemos três bases que têm uma relação muito importante entre elas. As bases de dados escolhidas são:

ESTADOS.sql, MUNICIPIOS.json e CEP.csv. Escolhemos estas três bases por existir relação entre elas. Uma dessas relações "um para muitos"pode ser visto no caso em que existe um estado que possui vários municípios e no caso de um município que contém vários CEP.

Para fazermos então uma consulta, utilizamos o atributo CEP que nos retorna todas as informação sobre um determinado CEP, isso é possível por causa da junção. Para usar essas bases precisamos de alguns tratamentos, sabendo que os dados no arquivo CSV são todos valores do tipo texto independentemente se eles são numéricos ou não, quando precisamos utilizar um valor inteiro de uma base SQL ou JSON e comparar os valores com uma base CSV, é necessário fazer uma conversão do tipo numérico para um tipo texto, pois todas as colunas do arquivo CSV sempre estarão com tipo texto, mesmo que uma coluna seja numérica. As bases possuem formato diferentes pois estão disponibilizadas desta forma. No caso desses arquivos cada arquivo representa uma tabela, para os tipos CSV e JSON o Apache Drill tem plugin para acessá-los diretamente, mas no caso do SQL não existe um plugin. Sendo assim, é necessário inserir os dados SQL em uma base MYSQL para então poder ter acesso no Apache Drill. Usamos um plugin chamado rdbms, o qual nos permitiu fazer a conexão com a base de dados no MYSQL.

Um subconjunto das bases de dados utilizados estão ilustradas nas Tabelas 4.1, 4.2, 4.3, sendo que nas entradas temos tabelas simplificadas com as principais colunas de cada base e algumas linhas de informações.

Os atributos destas bases são interpretados de duas formas diferentes, baseando-se no tipo de valor que possuem: valores tipo texto ou tipo numérico discreto (i.e tipo alfa-numérico); e valores tipo numérico contínuo (i.e tipo inteiro ou algum outro como long int).

Tabela 4.1: Dados da base estado.

codigo_uf	uf	nome
33	RJ	Rio de Janeiro
35	SP	São Paulo
41	PR	Paraná
42	SC	Santa Catarina
43	RS	Rio Grande do Sul

Tabela 4.2: Dados da base município.

codigo_ibge	nome	codigo_uf	ddd
4106902	Curitiba	41	41
4205407	Florianópolis	42	48
4314902	Porto Alegre	43	51
3548500	Santos	35	13
3306305	Volta Redonda	33	24

Tabela 4.3: Dados da base CEP.

cep	logradouro	bairro	cidade_ibge
80060140	Rua Doutor Faivre	Centro	4106902
1310000	Avenida Paulista	Bela Vista	3550308
88010000	Rua Felipe Schmidt	Centro	4205407

Como saída temos uma tabela que contém as informações da junção das três bases ilustradas acima. Esta saída é o resultado de uma consulta usando um CEP de uma rua e retornando as informações. No final temos como resultado o nome do estado, nome da cidade, o tipo de logradouro e nome do logradouro que está ilustrado na Tabela 4.4. Além de usar CEP na consulta, poderíamos usar outra informação como nome da cidade ou do estado. Observe que existe uma grande diferença no arquivo CSV, as colunas não possuem nome e não tem um tipo definido é simplesmente um array de colunas que é representado de como columns e column[0] é uma coluna na primeira posição da lista, por isso precisamos analisar os dados para então saber os nomes e os tipos de cada coluna. Como não é possível obter o nome de uma coluna dos arquivos CSV em uma consulta é necessário aplicar uma convenção nas colunas, como por exemplo: columns[0] as CEP. No caso do tipo da coluna foi feito a conversão para transformar no tipo correto **CAST(columns[1] as int)** sabendo que a segunda coluna é do tipo inteiro. Já as demais bases como arquivo JSON e SQL não possuem esse tipo de problema, pois essas já vem com nome e tipo das colunas definido. O SQL abaixo é montado baseado no Algoritmo 1.

```
select * from dfs.`/path/municipios.json` as m
join dfs.`/path/cep.csv` as c on c.columns[18]=CAST(m.codigo_ibge as char)
join rdbms.teste.estados as e on e.codigo_uf=m.codigo_uf
where c.columns[0]='80060140';
```

Tabela 4.4: Resultado de uma junção entre as Tabelas 4.1, 4.2 e 4.3.

cep	logradouro	bairro	cidade_nome	estado_nome
80060140	Rua Doutor Faivre	Centro	Curitiba	Paraná

4.3.2 Saída

Executamos uma consulta usando o SQL abaixo baseado no Algoritmo 1, nesta consulta nos utilizamos as bases de dados mostradas nas Tabelas 4.1, 4.2, 4.3, usamos a cláusula INNER JOIN para fazer a junção entre as tabelas citadas. O resultado desta consulta tem 1108505 linhas , neste caso optamos por simplificar esse resultado escolhendo algumas colunas e linhas na Tabela 4.5.

```
select t.columns[0] as CEP, t.columns[11] as Logradouro, d.nome as NomeCidade,
e.nome as NomeEstado, e.uf as Sigla
from dfs.`/path/cep.csv` as t
inner join dfs.`/path/municipios.json` as d
on t.columns[18]=CAST(d.codigo_ibge as char)
inner join rdbms.teste.estados as e on e.codigo_uf=d.codigo_uf limit 10;
```

O resultado obtido pela consulta do SQL acima é mostrado na Tabela 4.5, com cinco colunas: CEP e Lougradouro são duas coluna da Tabela 4.3 , NomeCidade é uma coluna da Tabela 4.2, NomeEstado e Sigla são duas colunas da Tabela 4.1. A tabela mostra um resultado simplificado de dez linhas de dados e nesta saída todas as colunas são do tipo texto. Foi feito alguns tratamentos no nome das colunas, na tabela cep existe cep e lougradouro originalmente, na saída utilizamos CEP e Lougradouro. Já no caso da tabela município existe uma coluna chamada "nome"e na tabela estados também existe uma coluna chamada "nome", para evitar conflito entre esses dois nomes fizemos uma convenção entre eles: NomeCidade e NomeEstado. Por fim, na base estado existe uf que é representado por Sigla.

Tabela 4.5: Resultado de uma junção entre as bases 4.1, 4.2 e 4.3.

CEP	Logradouro	NomeCidade	NomeEstado	Sigla
49070433	Rua Paulo Cardeal	Brusque	Santa Catarina	SC
01001000	Praca da Se	São Paulo	São Paulo	SP
60422110	Travessa Tres	Belém	Pará	PA
69001009	Margem Esquerda do Rio Amazonas	Manaus	Amazonas	AM
50010010	Rua Siqueira Campos	Recife	Pernambuco	PE
20010020	Rua Sao Jose	Rio de Janeiro	Rio de Janeiro	RJ
90002900	Rua Siqueira Campos, 1100	Porto Alegre	Rio Grande do Sul	RS
78005020	Rua Galdino Pimentel	Cuiabá	Mato Grosso	MT
57010001	Avenida Siqueira Campos	Maceió	Alagoas	AL
70002901	SBN Quadra 1 Bloco A 19o Andar	Brasília	Distrito Federal	DF

Neste experimento nos usamos somente a cláusula INNER JOIN, pois as bases de dados que nos usamos são de relações obrigatórias. Não existe diferença entre base de dados de relações obrigatórias usando qualquer tipo de JOIN tais como as cláusulas INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN e CROSS JOIN. No caso das tabelas usadas para fazer o experimento do nosso trabalho, a Tabela 4.1 possui uma relação obrigatório com a Tabela 4.2, e a Tabela 4.2 possui uma relação obrigatório com a Tabela 4.3, nelas não existem nenhuma linha que não está relacionada, por causa disso nenhuma das cláusulas citadas vai fazer diferença no resultado.

É possível observar que todos os resultados das tabelas 2.7 CROSS JOIN, 2.6 FULL JOIN, 2.5 RIGHT JOIN, 2.4 LEFT JOIN, 2.3 INNER JOIN, mostram claramente uma saída para cada uma das consultas executando as cláusulas INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN e CROSS JOIN nas tabelas como podemos ver por exemplo na Tabela 2.2. Essas consultas são abordadas no nosso trabalho, é extremamente importante saber quando usar cada uma dessas cláusulas.

4.4 CONCLUSÃO

Nesta seção definimos a arquitetura e descrevemos a implementação utilizada em nosso trabalho. Como o foco do nosso trabalho é unir bases de dados heterogêneas através de uma única consulta, criamos um programa em Java que utiliza a API do Apache Drill, o qual nos permite resolver o problema de junção entre bases de fontes e formatos diferentes. Alguns dos formatos testados incluem MYSQL, CSV e JSON. Além disso, apresentamos um estudo de caso para validar o sistema e testar sua eficácia, gerando assim uma saída para os resultados experimentais.

5 CONCLUSÃO

Neste trabalho abordamos um tema importante chamado integração de dados. Na fundamentação teórica explicamos o que é um banco de dados e seus principais conceitos. Nela é possível compreender todos os tipos de relações entre tabelas e os tipos de JOIN existentes. Na sequência foi apresentado uma visão geral sobre dados abertos e suas principais formas de armazenamento. Para finalizar, explicamos os conceitos integração de dados, seus principais desafios e em seguida algumas das tecnologias existentes que nos ajudaram no objetivo final.

Um subconjunto dos trabalhos relacionados com o tema integração de dados abertos foram explicados e comparados na Tabela 3.1, assim foi possível perceber se existiam características semelhante e quais as principais diferenças entre cada artigo.

Fizemos diversos testes para validar nosso trabalho junto aos conceitos de SQL, que podem ser resumidos como a junção de várias bases diferentes. Criamos as consultas usando dados abertos juntamente com a API do Apache Drill, a qual nos permite acessar os arquivos de formatos diferentes. Uma vez que rodamos o Apache Drill conseguimos enviar requisições com o SQL desejado. Para auxiliar criamos um programa em Java que permite obter o resultado de uma consulta. O retorno sobre as consultas se torna então uma única tabela. No exemplo das bases de dados de localização, inicialmente tínhamos três arquivos (que podem ser considerados tabelas) em formatos diferentes, ao final da consulta obtivemos um resultado com todas as colunas de todas as tabelas.

Por fim, um assunto bastante interessante da área de integração de dados abertos é o de corrompimento. Neste trabalho apenas alguns teste básicos de corrompimento foram feitos com o Apache Drill. Na maioria dos casos, a API simplesmente não retornava nenhum resultado, mesmo quando apenas uma entrada de uma base estava corrompida. Como possibilidade de trabalhos futuros, sugerimos como ideia a implementação de um sanitizador, que será uma etapa anterior a execução do Apache Drill. O papel do sanitizador será de simplesmente eliminar entradas corrompidas das bases, possivelmente evitando falhas na execução de consultas.

REFERÊNCIAS

Atiq, A. K. R. A. C. N. R. (2014). Challenges of data integration and interoperability in big data. https://ieeexplore.ieee.org/document/7023888. Acessado em 10/9/2021.

Atiq, A. K. R. A. C. N. R. (2015). Challenges of data integration and interoperability in big data. https://ieeexplore.ieee.org/document/7004486. Acessado em 10/9/2021.

Cury, T. (2015). Tipos de chaves no banco de dados relacional. http://thiagocury.eti.br/disciplinas/banco-de-dados-relacional/conceito-de-chaves-no-banco-de-dados-relacional.php. Acessado em 9/9/2021.

DADOS-ABERTOS (2021). Portal brasileiro de dados abertos. https://dados.gov.br/. Acessado em 12/9/2021.

DAMAS, L. (2007). Sql - structured query language, 6ª edição. https://integrada.minhabiblioteca.com.br/reader/books/9788521632450/epubcfi/6/40[%3Bvnd.vst.idref%3Dchapter07]!/4. Acessado em 10/9/2021.

del Fabro, M. D. (2018). Gestion de métadonnées utilisant tissage et transformation de modèle. https://tel.archives-ouvertes.fr/tel-00481520. Acessado em 28/10/2021.

Eike Schallehn, Kai-Uwe Sattler, G. S. (2003). Efficient similarity-based operations for data integration. https://www.researchgate.net/publication/222407045_Efficient_similarity-based_operations_for_data_integration. Acessado em 25/9/2021.

Fabro, C. (2020). O que é api e para que serve? https://www.techtudo.com.br/listas/2020/06/o-que-e-api-e-para-que-serve-cinco-perguntas-e-respostas.ghtml. Acessado em 9/9/2021.

FAIS, P. R. M. (2009). Tipos de relacionamentos em um banco de dados relacional. https://www.rmfais.com/rmfais/artigos/table.php?_codigo=6. Acessado em 9/9/2021.

Foundation, T. A. S. (2012-2020). Drill introduction. https://drill.apache.org/docs/drill-introduction/. Acessado em 9/9/2021.

Gehrke, R. R. J. (2007). Sistemas de gerenciamento de banco de dados, 3ª edição. https://integrada.minhabiblioteca.com.br/reader/books/9788563308771/pageid/0. Acessado em 27/10/2021.

Hariharan, B. B. A. T. R. S. (2014). Data integration progression in large data source using mapping affinity. https://ieeexplore.ieee.org/document/7004486. Acessado em 10/9/2021.

Kaggle (2021). Take the annual kaggle ml and data science survey! https://www.kaggle.com/. Acessado em 12/9/2021.

Lenzerini, M. (2002). Data integration: a theoretical perspective. https://dl.acm.org/doi/pdf/10.1145/543613.543644. Acessado em 10/9/2021.

Microsoft (2021). Acesso sql: conceitos básicos, vocabulário e sintaxe. https://support.microsoft.com/pt-br/office/acesso-sql-conceitos-b%C3%A1sicos-vocabul%C3%A1rio-e-sintaxe-444d0303-cde1-424e-9a74-e8dc3e460671. Acessado em 29/11/2021.

Miller, R. J. (2018). Open data integration. https://dl.acm.org/doi/pdf/10.14778/3229863.3240491?casa_token=Rh-pT7MzCTQAAAAA:FJucMim7PijgUFmmef0w3NIs_0ybj_lp1EglvctmmsqxSbeN3InADNkaANN-qPlR_8xmjk5rRqWrIA. Acessado em 10/9/2021.

Office, C. (2012). Open data white paper. https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/78946/CM8353_acc.pdf. Acessado em 12/9/2021.

Oracle (2014). O que é sql (structured query language, linguagem de consulta estruturada)? https://www.oracle.com/br/database/what-is-database/#link2. Acessado em 10/9/2021.

Pichetti, Roni, F. e. a. (2021). Banco de dados, 1ª edição. https://integrada.minhabiblioteca.com.br/reader/books/9786556900186/pageid/126. Acessado em 27/12/2021.

Pisa, P. (2012). O que é e como usar o mysql? https://www.techtudo.com.br/artigos/noticia/2012/04/o-que-e-e-como-usar-o-mysql.html. Acessado em 9/9/2021.

Seiji Isotani, l. I. B. (2015). Dados abertos conectados. http://pgcl.uenf.br/arquivos/dadosabertosconectados_011120181613.pdf. Acessado em 07/01/2022.

Setzer, W. S. (2005). Bancos de dados., 1ª edição. https://integrada.minhabiblioteca.com.br/reader/books/9788521216520/pageid/0. Acessado em 24/11/2021.

Shafranovich, Y. (2005). Comma-separated values. https://datatracker.ietf.org/doc/html/rfc4180. Acessado em 25/9/2021.

SILBERSCHATZ, A. (2020). Sistema de banco de dados, 7ª edição. https://integrada.minhabiblioteca.com.br/reader/books/9788595157552/epubcfi/6/24%5B%3Bvnd.vst.idref%3Dchapter1%5D!/4/130/1:164%5Bssa%2Cmen%5D. Acessado em 27/10/2021.

Williams Alcantara, J. B. (2015). Desafios no uso de dados abertos conectados na educação brasileira. https://sol.sbc.org.br/index.php/desafie/article/download/10036/9918. Acessado em 07/01/2022.