UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Victor Hugo Eustáquio Lopes

Desenvolvimento de uma Ferramenta ETL para conversão de dados semiestruturados e estruturados em JSON para o modelo relacional

Uberlândia, Brasil

2023

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Victor Hugo Eustáquio Lopes

Desenvolvimento de uma Ferramenta ETL para conversão de dados semiestruturados e estruturados em JSON para o modelo relacional

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Ciência da Computação.

Orientador: Prof. Humberto Luiz Razente

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2023

Victor Hugo Eustáquio Lopes

Desenvolvimento de uma Ferramenta ETL para conversão de dados semiestruturados e estruturados em JSON para o modelo relacional

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Ciência da Computação.

Trabalho aprovado. Uberlândia, Brasil, 01 de setembro de 2023:

Prof. Humberto Luiz Razente
Orientador

Profª Maria Adriana Vidigal de Lima

Prof. Ilmério Reis da Silva

Uberlândia, Brasil
2023

Dedico aos meus pais que sempre incentivaram, apoiaram e deram suporte naquilo que precisei! Dedico também aos professores que sempre se dispuseram a passar seus conhecimentos

Resumo

O mundo digital não para de gerar informações sobre o que as pessoas estão fazendo e consumindo, desde o interesses de compras, vídeos de entretenimento, cursos sendo comprados e criados, dentre outras infinidades de coisas. Os ambientes de tecnologia da informação contam atualmente com dados armazenados em Sistemas de Gerenciamento de Bancos de Dados Relacionais e com dados armazenados em Sistemas NoSQL. Com isso, muitas empresas utilizam esses dados a seu favor para alavancar seus negócios e resultados de forma precisa, mas é necessário saber quais dados são relevantes para realizar a análise, porém, estes podem vir de um banco de dados diferente do utilizado pela empresa.

Por isso, pode-se fazer necessária conversão dos dados de um modelo para o outro. Neste trabalho é apresentada uma ferramenta para a conversão de documentos em formato JSON, que é amplamente utilizado em diversos sistemas, para o modelo relacional. Para isso é preciso a análise dos documentos e seus respectivos pares de chave/valor para o mapeamento em tabelas, por meio da criação dos atributos e dos relacionamentos entre as tabelas, permitindo a extração, transformação e carga de um banco de dados de documentos JSON para um banco de dados relacional, permitindo a manipulação e execução de consultas expressas na linguagem SQL.

Palavras-chave: Modelo relacional, ferramenta ETL, JSON.

Lista de ilustrações

Lista de abreviaturas e siglas

JSON	JavaScript Object Notation
IBM	International Business Machines
ETL	Extract, Transform, Load
SAP	System Analysis Program Development
SQL	Structured Query Language
SGBD	Sistema Gerenciador de Banco de Dados
XML	Extensible Markup Language
HTML	HyperText Markup Language
PK	Primary Key
ACID	Atomicidade, Consistencia, Isolamento, Durabilidade
WEB	World Wide Web
BLOB	Binary large object
NoSQL	No Structured Query Language
CPF	Cadastro Pessoa Física
BI	Business Intelligence
OLAP	Online Analytical Processing
DBMS	Data Base Manage System
HTTP	Hypertext Transfer Protocol
API	Application Programming Interface

1 Introdução

Muitos usuários que utilizam a internet não percebem que têm os seus dados guardados pelas empresas em bancos de dados. Segundo Elmasri e Navathe (2011), um banco de dados é uma coleção de dados relacionados que são informações que condizem com a realidade e são guardados com um propósito específico. Esse banco é gerenciado por um SGBD (Sistema de Gerenciamento de Banco de Dados), com ele é possível realizar operações de manipulação de dados, como a inclusão, atualização, remoção e consultas para atender aos requisitos das aplicações.

Hoje no mundo globalizado e digital, uma grande quantidade de dados são gerados a todo o momento. Segundo Bernard Marr (2018), em 2018 foi estimado que aproximadamente 2,5 quintilhões de *bytes* foram gerados todos os dias. Muitos destes dados podem ser utilizados para realizar análises e direcionar campanhas e produtos de forma mais assertiva ou prever acontecimentos. Entretanto, muitos podem não ter significado ou não se consegue retirar algum valor considerado útil. Com o avanço das tecnologias de desenvolvimento de sistemas, o arquivo JSON surgiu como alternativa ao XML, e vem sendo amplamente utilizado por ser uma forma simples e leve de armazenar dados, facilitando o *parsing* e entendimento da estrutura dos mesmos, de modo que os algoritmos podem fazer a sua interpretação.

Os arquivos XML e JSON permitem o armazenamento de dados estruturados, equivalentes ao modelo relacional, mas também podem armazenar dados semiestruturados ou não estruturados. O modelo relacional tem por objetivo eliminar a redundância de dados com a criação de esquemas normalizados. Para se obter um esquema normalizado a partir de dados semi ou não estruturados, é preciso realizar a análise e transformação dos dados. A não existência de dados redundantes é importante para garantir o desempenho de um SGBD, facilitando as análises dos dados. É importante destacar que é possível armazenar arquivos multimídias em SGBDs, como XML, JSON, áudios, vídeos, textos, documentos, entre outros, por meio de atributos do tipo BLOB (*binary large object*), porém, para muitas tarefas, a sua manipulação depende de suporte específico do SGBD para o tipo armazenado.

Portanto, como o JSON é muito utilizado nas aplicações *WEB* e *mobile*, sendo as principais fontes de geração de informação, muitas vezes se faz necessário realizar um tratamento destes dados, pois muitos destes podem possuir valores nulos, atributos multivalorados, listas, agregações, aninhamento de estruturas, sendo naturais no modelo de documentos em sistemas NoSQL, mas que demandam tratamento para transformação e carga num banco de dados relacional, com a transformação do modelo para ao menos a

terceira forma normal (3FN).

Para retirar algum significado destes dados, é preciso realizar diversos procedimentos, uma vez que muitos podem não estar preenchidos ou precisam ser combinados com outros dados para ter um sentido. Optar pela remoção de colunas que possuem muitos valores nulos, se estes não possuírem algum significado quando vazios, caso contrário pode-se gerar valores para substituí-los, em muitos casos sendo necessário realizar testes e estudos sobre qual melhor método para tal preenchimento. Para isso existem diversos métodos estatísticos, como o preenchimento de dados utilizando a média ou mediana, a cópia de valores de outros dados que são similares, valor selecionado aleatoriamente dentre os existentes, dentre outros métodos. Esta atividade é denominada ETL (extração, transformação e carga), no qual é realizada a extração de dados, podendo ser de um banco de dados, junção de vários ou através de pesquisas ou observações, e guarda-los. O tratamento consiste em deixar estes dados da forma em que faça mais sentido tê-los, e o carregamento deles para um banco de dados relacional (*data warehouse*) para que posteriormente possa ser utilizado para a realização de consultas analíticas.

Segundo Aftab et al. (2020), o crescimento do uso de bancos de dados NoSQL ocorreu com foco de aumentar a escalabilidade, permitindo disponibilidade e tolerância ao particionamento, além da necessidade de não ficar restrito ao modelo relacional, permitindo gerenciar dados não estruturados e semiestruturados para que seja possível ter maior flexibilidade em lidar com dados massivos. Por outro lado, há aplicações nas quais há o requisito das transações ACID disponibilizadas pelos Sistemas de Gerenciamento de Bancos de Dados (SGBDs), que restringem a escalabilidade desses sistemas. As propriedades ACID de acordo com Elmasri e Navathe (2011) são:

- atomicidade: conceito que envolve duas ou mais informações, em que ou a tarefa será totalmente executada ou não será executada, garantindo assim, que sejam atômicas. Desta forma, é garantido que se uma das operações falhar, todas as operações feitas dentro de uma transação serão desfeitas.
- consistência: após a execução de uma transação, o novo estado em que se encontra o banco de dados é consistente após a transação.
- isolamento: garante que cada transação pareça estar sendo executada isoladamente, embora centenas de transações possam estar sendo executadas de modo simultâneo.
- durabilidade: os dados são mantidos mesmo que ocorra uma falha do sistema.

Segundo Aftab et al. (2020), ferramentas ETL são comumente utilizadas para extrair dados, e um dos exemplos apresentados é o *Talented Open Studio*, no qual é necessário o mapeamento manual dos esquemas, o que se torna muito difícil com sistemas NoSQL, pois estes permitem a manipulação de dados semiestruturados ou não-estruturados.

Portanto, como alguns bancos de dados NoSQL armazenam os seus dados em formato JSON, devido a sua flexibilidade, na hora que as empresas vão realizar análise desses dados para obter resposta sobre os seus processos ou até onde investir, precisam utilizar ferramentas que consomem dados em formato relacional. Desta forma, é necessário fazer a transformação destes arquivos em JSON para o SQL. Portanto, esse projeto tem como intuito fazer essa transformação.

Cada um destes conceitos citados anteriormente, como ferramentas ETL, *data warehouses*, banco de dados relacionais, banco de dados não relacionais e assim por diante, serão apresentados nos capítulos seguintes.

1.1 Objetivos

O principal objetivo deste trabalho foi o estudo e desenvolvimento de um mecanismo para extração, transformação e carga de dados semiestruturados em JSON para o modelo relacional. Para tanto, foi criado um *parser* de JSON capaz de receber uma coleção de documentos JSON e convertê-los para o modelo relacional (criando tabelas, seus atributos e relacionamentos), podendo realizar alterações necessárias no modelo, caso sejam adicionados novos documentos JSON. Para alcançar este objetivo, foram necessários:

- A criação de uma função em que o usuário consiga receber os dados.
- A criação de um função que lê e converte o arquivo no formato JSON utilizando a ferramenta ETL para o SGBD relacional.
- A construção de um arquivo JSON com dados para aplicar os sistemas.
- Ao final da execução, é esperado que se tenha instruções SQL para construção das tabelas com seus respectivos atributos e instruções SQL para inclusão dos dados do arquivo passado como parâmetro.
- Realização de testes para validação dos resultados obtidos, verificando se o modelo resultante é equivalente ao não relacional.

No final, o resultado obtido foi de um algoritmo que recebe um arquivo JSON de entrada, salva localmente e faz a leitura do mesmo verificando as estruturas existentes e verificando quais tabelas, atributos e relacionamentos existentes nos arquivos passados. Quando o algoritmo termina este processo de leitura é exportado um arquivo com as instruções SQL para criação e povoação das tabelas.

1.2 Metodologia

Para realização do projeto, foi necessário realizar a pesquisa bibliográfica para levantamento do estado da arte sobre assuntos relacionados ao modelo relacional, ferramentas ETL, data warehouses e classificação dos sistemas de gerenciamento de bancos de dados. Além disso, foi necessário o estudo do formato JSON, com o fim de entender como manipulá-lo e quais tipos de dados são aceitos. Essa pesquisa abordou sobre como é realizado o *parsing*, como é utilizada e funciona uma ferramenta ETL, sendo estudada diferentes técnicas para a transformação dos dados extraídos, e como analisá-los. Além disso, entender as estruturas de dados do banco relacional e como devem ser construídos, como funciona sua estrutura, diferença entre os produtos existentes hoje. Como deve ser interpretado as instâncias dos arquivos JSONs para construção do banco e tratamento de possíveis problemas com duplicação de valor, valores nulos, além de conhecer como manipulá-lo. Após o estudo, foi necessária escolha de ferramentas e linguagens de programação para realização do desenvolvimento da ferramenta proposta e quais técnicas de tratamento dos dados seriam usadas. Concluído isto, foi iniciada a implementação do sistema, seguido de diversos testes para avaliação dos resultados, nos quais foram propostos e implementados diversas melhorias ou correções.

2 Revisão Bibliográfica

De acordo com o Schreiner, Duarte e Mello (2020), a quantidade massiva de dados que são gerados por segundo trouxe muitos desafios para as aplicações centrada em dados, por exemplo, ter que lidar com dados heterogêneos, grande de volume de dados e o rápido crescimento da quantidade de dados gerados. Além disso, muitos desses dados não possuem um esquema, uma vez que são estruturados de forma flexível e podem ser heterogêneos. Devido a isso, os bancos NoSQL foram propostos para conseguir solucionar esses problemas que os bancos de dados relacionais, que até então eram muito utilizados, não conseguiam resolver, uma vez que os bancos NoSQL tem a capacidade de oferecer alta escalabilidade, armazenamento de dados sem esquema e alta disponibilidade. Alguns exemplos citados no artigo de aplicações que possuem essas características são as redes sociais, redes de sensores e assistência médica.

Para compreender todo o trabalho aqui realizado, é importante o entendimento sobre o que é um banco de dados, como funciona, quais tipos existem, e o que difere entre eles. Além disso, é necessário conhecer a ferramenta ETL, mostrando para que ela serve, como deve ser utilizada, qual seu objetivo, e por último, mas não menos importante, é necessário conhecer o formato de dados JSON, como ele é organizado e como utilizá-lo.

2.1 Bancos de Dados

De acordo com Harrington (2016), um conceito fundamental do banco de dados é a existência de entidades para as quais deseja-se a persistência e como essas entidades se relacionam umas com as outras. Entretanto, para ser considerado de fato um banco de dados, não se deve ter apenas os dados armazenados em um lugar, mas é preciso que se tenha guardado também a forma em que cada um dos dados estão relacionados. Um exemplo é a possibilidade de relacionar os clientes com os pedidos que eles fazem em uma empresa, bem como relacionar com o estoque destes itens.

Em Rob e Coronel (2011) é apresentada uma comparação do que são informações e dados. Para ele a definição de dado é dado como sendo algo que não foi processado para retirar um significado, ao passo que as informações são o resultado do processamento dos dados, tendo agora um significado sobre eles. Este processamento pode ser apenas uma organização dos dados e identificação de padrões, até coisas complexas como a utilização de modelos estatísticos. Além disso, para conseguir saber o significado dos dados, é preciso saber qual o contexto em que está o dado que nos foi entregue. Por exemplo, o armazenamento de uma temperatura igual a 105º necessita da informação sobre a sua unidade, Fahrenheit ou Celsius, e a interpretação pode ser diferente caso esteja se refe-

rindo a uma máquina, a um corpo ou a temperatura atmosférica. Define-se então que um banco de dados é uma estrutura computacional compartilhada e integrada que armazena um conjunto de dados brutos e metadados, ou seja, dados sobre dados.

Segundo Elmasri e Navathe (2011), os bancos de dados mudaram muito desde o seu surgimento, que ocorreu em 1960, vários tipos de organizações dos dados foram criadas, sendo cada um é mais adequado para um determinado tipo de uso. Os primeiros bancos eram hierárquicos era baseado no modelo de árvore, permitindo apenas um relacionamento um para muitos, ou de rede, esse já permitia maior quantidade de relacionamentos, ambos eram simples, mas inflexíveis. Nos anos 1980, os bancos de dados relacionais e os bancos de dados orientados a objetos se tornaram muito populares, mas mais recentemente os bancos NoSQL está se tornando mais usados por causa da internet e pela necessidade de maior velocidade de processamento de dados não estruturados. Hoje já está começando a ser utilizado bancos de dados em nuvem. Existem também os bancos de dados autônomos, estes já são bancos baseados em nuvem e utilizam *machine learning* para automatizar o ajuste de banco de dados, segurança, backups, dentre outras tarefas.

2.1.1 Normalização de dados

Um conceito importante na área de banco de dados é a normalização de dados. De acordo com Elmasri e Navathe (2011), ela é um processo que visa a preservação da informação, incluindo os tipos de dados, entidades, relacionamentos, e a minimização da redundância, ou seja, diminuindo o armazenamento redundante da mesma informação e a necessidade de múltiplas atualizações para manter a consistência entre elas. Este processo foi proposto por Codd, e é composto por três principais formas normais mais usadas, sendo chamadas de primeira, segunda e terceira forma normal. Entretanto, existem outras que tratam casos particulares. Além disso, são apresentadas quatro diretrizes informais que podem ser utilizadas para medir a qualidade de um projeto do esquema da relação. A Figura 1 apresenta 2 tabelas e respectivas dependências funcionais que serão utilizadas para exemplificar os problemas explicados.

A primeira é garantir que a semântica dos atributos é clara no esquema, ou seja, não se deve combinar atributos de vários tipos de entidade e de relacionamento em uma única relação, pois podem existir ambiguidades semânticas. Quando se armazena muitas relações em um lugar pode ocorrer algumas anomalias, podendo elas ser de inserção, exclusão e modificação. Elas foram identificadas por Codd para justificar a necessidade de normalização das relações. Já a segunda diretriz diz para projetar esquemas de relações que não possuam as anomalias citadas anteriormente e as que vão ser explicadas em seguida.

As anomalias de inserção podem ser diferenciadas em dois tipos, sendo a primeira a de coerência, em que para inserir um novo dado pode acontecer de inserir dados incorretos.

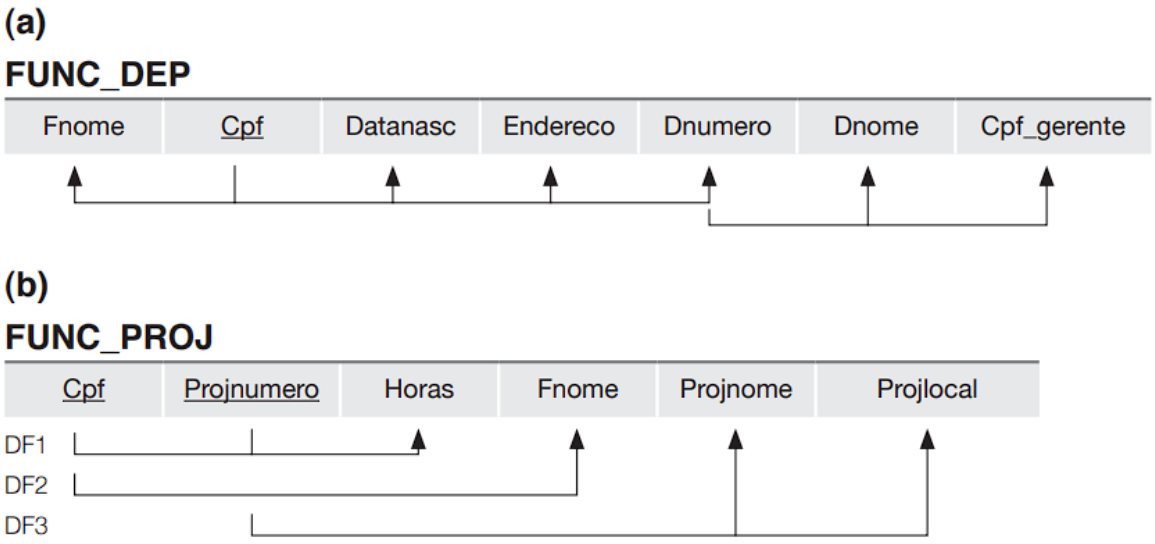

Figura 1 – Exemplo de uma tabela para representar as anomalias Elmasri e Navathe (2011)

Um exemplo em Elmasri e Navathe (2011) é do caso de uma tabela que contém dados de um profissional que trabalha em um departamento, representada por FUNC_DEP na Figura 1, e é necessário inserir o nome e o CPF do gerente. Caso algo esteja diferente na inserção em relação ao que já existe, ela irá gerar uma incoerência, pois em outras tuplas estará um valor e na que foi inserida outro, não sabendo assim qual o correto. Por exemplo, caso tenha o funcionário João no departamento de número 1 com nome secretaria, e outro funcionário com nome Maria com departamento também de número 1 mas o nome está como financeiro teremos uma incoerência, pois não se sabe se o departamento de número 1 se chama financeiro ou secretaria.

Além disso, existe a violação de integridade, por exemplo, quando é inserido um departamento, é necessário um gerente, mas como não tem funcionário ainda cadastrado este campo precisa receber NULL, o que viola a entidade já que o CPF do gerente é a chave primária.

O problema das anomalias de exclusão estão relacionadas à segunda situação de anomalia de inserção que vamos discutir. Se exclui de uma tabela, no qual possui os dados de um funcionário e os dados departamento, uma tupla de funcionário que representa o último funcionário trabalhando para determinado departamento, a informação referente a esse departamento se perde do banco de dados, uma vez que esta tabela possui os dados de departamento. O correto seria separar em duas entidades e passar o número do departamento para o funcionário, desta forma não se perde nenhuma informação que não desejava ser removida. Na figura 1 é possível obter isso, já que FUNC_DEP possui dados do funcionário e do departamento, caso exista apenas um funcionário em um departamento

e ele seja excluído, as informações com nome e número dele também são perdidas.

A anomalia de modificação ocorre quando uma tabela possui valores que se repetem em outras tuplas e elas informam a mesma coisa, caso seja mudada em uma, deve ser mudada em todas. Seguindo o exemplo dado na anomalia de exclusão, se o nome de um departamento mudar em uma tupla, todas as outras que são do mesmo departamento devem mudar também, caso não ocorra se obterá uma inconsistência e não se saberá depois qual informação é a correta ou mais atualizada. Na figura 1 temos o nome do departamento, caso ele seja atualizado em uma instância, então deve ser atualizado em todas as outras com o mesmo número. Por exemplo, caso tenha o funcionário João no departamento de número 1 com nome secretaria, e outro funcionário com nome Maria com departamento também de número 1, se atualizar em Maria o nome do departamento para financeiro, é necessário mudar em João também já que ambos estão no mesmo departamento.

A diretriz 3 fala para evitar colocar atributos em uma relação cujos valores podem ser nulos com frequência. Quando isso acontece pode haver desperdício de espaço e também ocorrer problemas com o significado dos atributos. Além disso, eles podem ter diversas interpretações, como podendo ser o atributo não se aplica a tupla, ou o atributo não é conhecido, ou o atributo ainda não foi registrado, mas sabe-se o seu valor. Elmasri e Navathe (2011) cita então que deve ser utilizado de modo eficaz o espaço de armazenamento e evitar juntar relações que vão ocasionar valores nulos, isso deve ser analisado para determinar a inclusão ou não de colunas em uma relação, ou se serão relações separadas.

A quarta diretriz fala sobre projetar esquemas de relação em que não se consiga gerar nenhuma tupla que seja falsa. Desta forma, deve-se evitar relações de atributos correspondentes que não sejam combinações. Isso quer dizer que deve-se garantir que os atributos analisados, são atributos chave primária e chave estrangeira, pois quando não são há mais risco da geração de tuplas incorretas.

As dependências funcionais Elmasri e Navathe (2011) são restrições entre dois conjuntos de atributos do banco de dados. Por exemplo, as seguintes dependências funcionais:

- $CPF \rightarrow Fnome$
- $Projnumero \rightarrow \{Projnome, Projlocal\}$
- $\{CPF, Projnumero\} \rightarrow Horas$

mostram que com o cadastro de pessoa física (CPF) é possível determinar qual o nome de um funcionário. Na segunda, quando obtém-se o número de um projeto é possível determinar qual o nome e o local do projeto. Já a terceira, mostra que quando se tem o CPF de uma funcionário e o número de um projeto é possível determinar a quantidade de horas que este funcionário dedicou em um projeto.

Com isso, é possível ver que essas relações são muito importantes para encontrar determinadas informações que se relacionam. Entretanto, anomalias podem surgir e deve-se sempre verificar se nenhuma delas ocorre. Um conceito importante na área de banco de dados é a normalização de dados. Ela é um processo que tem como objetivo é a preservação da informação, incluindo os tipos de dados, entidade e relacionamento, e a minimização da redundância que por sua vez serve para diminuir o armazenamento redundante da mesma informação e a necessidade de múltiplas atualizações para manter a consistência entre as várias informações repetidas. Este processo foi proposto por Codd, e é composto por três principais formas normais mais usadas, sendo chamadas de primeira, segunda e terceira forma normal. Existem outras que tratam de anomalias específicas Elmasri e Navathe (2011). O processo de normalização consiste na análise de um esquema de relações e uma série de testes e decomposições para garantir que ela está satisfazendo as regras.

A normalização de dados pode ser vista como a análise dos esquemas de relação levando em consideração as suas dependências funcionais e chave primárias para conseguir realizar a minimização da redundância e anomalias já citadas anteriormente. Quando é encontrada uma relação que não satisfaz alguma condição da forma normal, ela pode ser dividida em esquemas de relações menores. Este processo de análise e desmembramento pode ser feito com todo o banco de dados relacional até um certo nível desejado. O grau de formalização de uma relação é indicado pela condição formal mais alta que ela atende e ao realizar o processo de decomposição, é necessário garantir que não vá ocorrer nenhuma geração de tupla falsa e ocorrer a preservação de dependência funcional Elmasri e Navathe (2011).

A primeira forma normal tem como objetivo a remoção de atributos multivalorados ou compostos. Um esquema de relações que está nesta forma normal deve conter apenas atributos simples e indivisíveis, ou seja, atributos que não são possíveis de serem divididos em mais de um e não recebem mais de valor (monovalorados). Para isso, uma forma para solucionar este problema é a criação de uma nova relação (tabela) que possui a chave primária da tabela existente e o valor de um atributo.

Portanto, caso uma tabela possua um atributo multivalorado, esse atributo deve ser separado para uma nova tabela contendo uma referência à tupla de origem (uma referência à chave primária) e um valor monovalorado, ambos compondo a chave primária nesta nova tabela. Caso seja um atributo composto, o mesmo deve ser decomposto em atributos monovalorados na própria tabela.

Na Figura 2 é possível ver que os atributos `Projnumero` e `Horas` possuem mais de um valor na primeira tabela. Então foi criada as tabelas `FUNC_PROJ1` e `FUNC_PROJ2` com o atributo CPF fazendo referência a `FUNC_PROJ1`.

A segunda forma normal diz que cada atributo não chave deve depender da chave primária por completo, ou seja, caso exista uma chave formada por mais de um atributo

(b)

FUNC_PROJ

Cpf	Fnome	Projnumero	Horas
12345678966	Silva, João B.	1	32,5
		2	7,5
66688444476	Lima, Ronaldo K.	3	40,0
45345345376	Leite, Joice A.	1	20,0
		2	20,0
33344555587	Wong, Fernando T.	2	10,0
		3	10,0
		10	10,0
		20	10,0
99988777767	Zelaya, Alice J.	30	30,0
		10	10,0
98798798733	Pereira, André V.	10	35,0
		30	5,0
98765432168	Souza, Jennifer S.	30	20,0
		20	15,0
88866555576	Brito, Jorge E.	20	NULL

(c)

FUNC_PROJ1

Cpf	Fnome

FUNC_PROJ2

Cpf	Projnumero	Horas

Figura 2 – Exemplo de uma tabela antes e depois de aplicar a primeira forma normal Elmasri e Navathe (2011)

deve-se conseguir definir qual tupla deseja-se buscar utilizando todos os atributos-chave e não apenas parte dela. Isso quer dizer que deve existir uma dependência funcional total. Se uma tabela possuir apenas um atributo como chave primária, e ela atender à primeira forma normal, então ela já está na segunda forma normal. Para conferir se um esquema está na segunda forma normal devem ser testadas as dependências funcionais. Entretanto, caso a chave primária seja formada por apenas um atributo, o teste não precisa ser aplicado. Por exemplo, caso se tenha uma tabela com chave primária formada pelos atributos CPF e Nprojeto e se algum atributo puder ser definido utilizando apenas CPF ou Nprojeto, então ela não está na segunda forma, pois depende de apenas parte da chave.

Na Figura 3 é possível observar que os atributos `Fnome` depende apenas de Cpf,

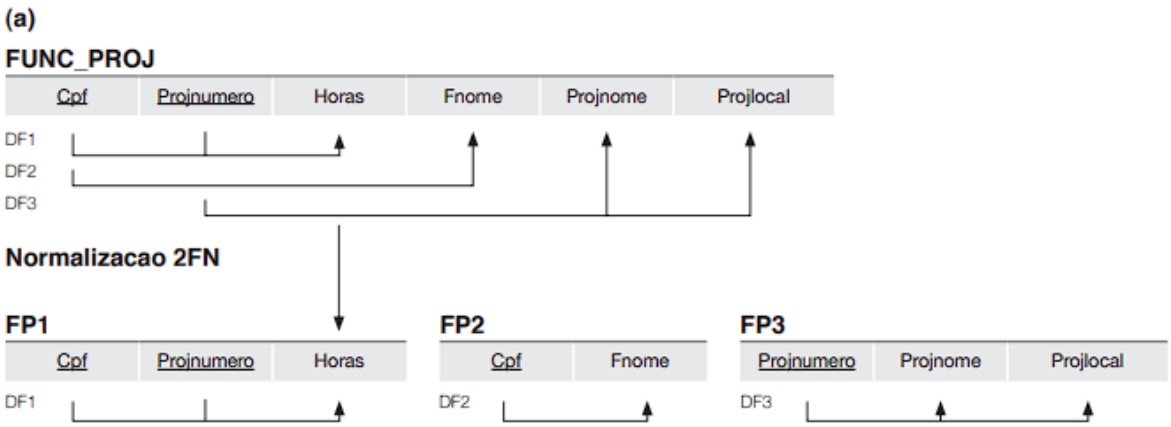

Figura 3 – Exemplo de uma tabela antes e depois de aplicar a segunda forma normal Elmasri e Navathe (2011)

já `Projnome` e Projlocal depende apenas de `Projnumero`, então foi criada as tabelas `FP1` com apenas Horas, Cpf, e Projnumero, FP2 com Cpf e Fnome e FP3 com Projnumero, Projnome e Projlocal. Desta forma foram separados os atributos com depêndencia parcial das chaves primárias garantindo que dependem integralmente da mesma. Com isso garantimos que FP1, FP2 e FP3 estão na segunda normal.

A terceira forma normal baseia-se no conceito de dependência transitiva. A dependência transitiva ocorre quando um atributo não chave depende de outro atributo não chave. Uma relação está na terceira forma normal se estiver na segunda forma normal e se não existirem atributos não chave que sejam dependentes de outros atributos não chave. Por exemplo, na Figura 4 há uma tabela chamada FUNC_DEP que contém os atributos *Fnome, Cpf, Datanasc, Endereco, Dnumero, Dnome, CpfGerente*, sendo o *Cpf* a chave primária, e a dependência funcional $Dnumero \rightarrow \{Dnome, CpfGerente\}$. Portanto, essa tabela não está na terceira forma normal.

Portanto, deve-se desmembrar esta tabela em duas, em que uma será definida por *Fnome, Cpf, Datanasc, Endereco, Dnumero* e uma nova tabela definida por *Dnumero, Dnome, CpfGerente*. Além disso, Dnumero em FUNCDEP será chave estrangeira da nova tabela, e assim todos os atributos da nova tabela dependem da chave primária, e não de um atributo simples, como ocorria antes em uma única tabela.

2.1.2 Bancos de dados não relacionais

Nas últimas quatro décadas, os bancos de dados relacionais têm sido amplamente utilizados em sistemas de informações. Todavia, na última década houve um aumento na manipulação de dados heterogêneos, sem esquema definido. Deste modo, os bancos NoSQL foram criados com a finalidade de ter o suporte para oferecer alta escalabilidade e elasticidade, sendo possível manipular conjuntos de dados que podem aumentar seu

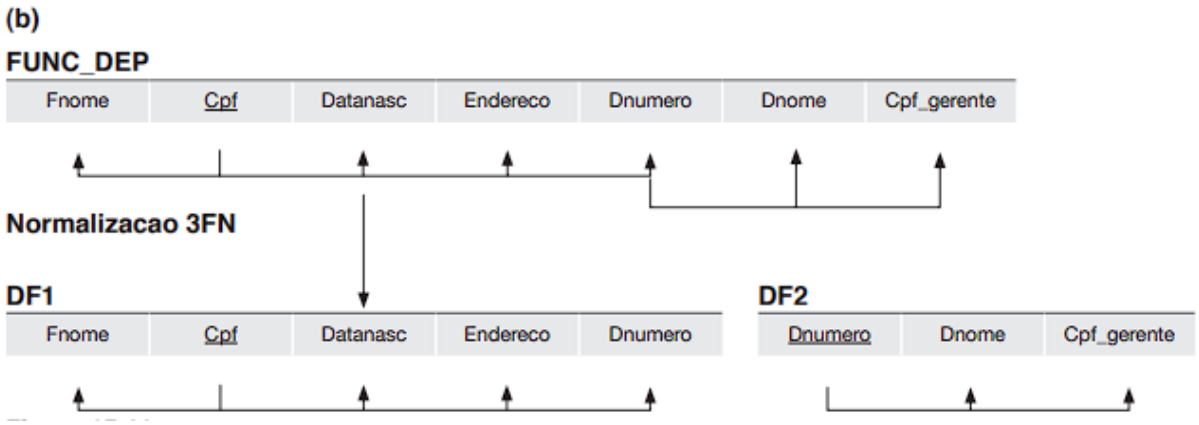

Figura 4 – Exemplo de uma tabela antes e depois de aplicar a terceira forma normal Elmasri e Navathe (2011)

tamanho de forma muito rápida, isso só é possível porque ocorre a flexibilização de algumas regras de consistência de dados. Os principais benefícios que ele possui são:

- Flexibilidade: com os bancos *NoSql*, é possível armazenar dados sem a forma rígida de uma estrutura pré-definida, sendo fácil lidar com dados semiestruturados , ou desestruturados;
- Escalabilidade: foram projetados para escalar de forma horizontal, ou seja, utilizam-se de clusters de computadores (sistemas distribuídos);

Em Pokorny (2011) são apresentados os principais tipos de bancos de dados NoSQL. As formas que os dados podem ser armazenados pode ser:

- Chave-valor: neste tipo, cada informação possui uma chave e um campo para valor, desta forma é possível acessar informações sem a necessidade de fazer consultas complexas, pois todas as informações que se relacionam entre si já estão juntas. Na Figura 5 é mostrado um exemplo deste tipo, em que do lado esquerdo possui o valor chave e do lado direito as informações referentes a essa chave.
- Orientado a documento: os dados são armazenados em documentos que são objetos, possuem um identificador e várias informações, podendo elas ter diversos tipos. Este tipo de banco é uma boa opção para armazenar dados não estruturados. Na Figura 6 é mostrado um exemplo de como pode ser visto sua organização. Do lado esquerdo é possível observar um documento e do lado direito outros, no qual ele possui relação com o da esquerda.
- Orientado à coluna: este modelo é o oposto do relacional que armazena conjuntos de dados somente em uma linha. Neste modelo, é armazenado os dados em várias linhas da tabela.

Key	Value
Book Title	Business Intelligence and Analytics: Systems for Decision Support
Author (set)	Ramesh Sharda
	Dursun Delen
	Efraim Turban
Publication Date	2015
Edition	10th
Publisher	Pearson
...	...

Figura 5 – Exemplo de um arquivo JSON Mason (2015)

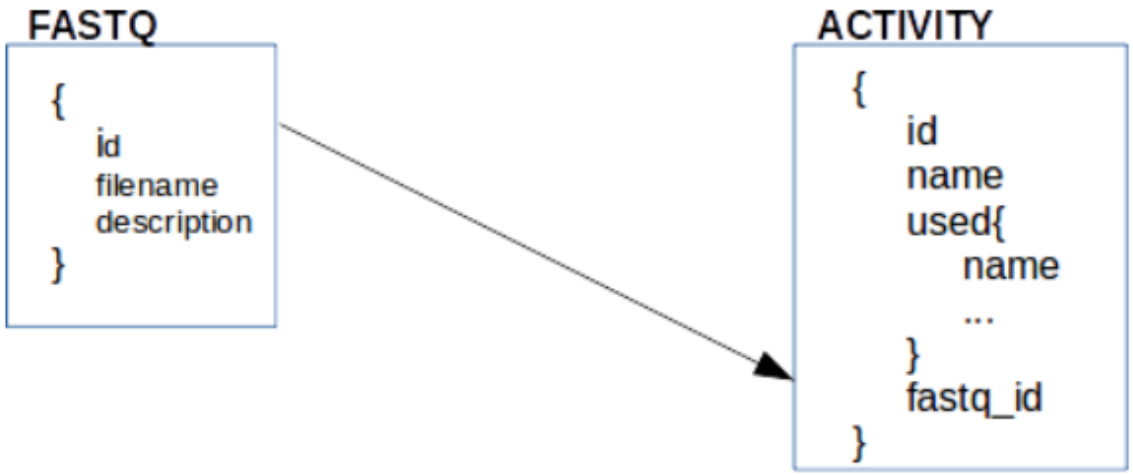

Figura 6 – Exemplo de um banco não relacional orientado a documento Vera et al. (2015)

EmpID	Salary	Designation
100	10,000	Clerk
200	20,000	Assistant Manager
300	30,000	Manager
400	40,000	Zonal Head

Figura 7 – Exemplo de um banco não relacional orientado à coluna Sharma e Dave (2012)

- Grafos: neste tipo, os dados junto com seus atributos são guardados nos nós, já as arestas é a relação entre cada um dos dados. Ele é indicado quando se deseja fazer

buscas complexas devido ao seu ganho de desempenho. Na Figura 8 é mostrado um exemplo deste tipo de dado, nela é possível observar cada um desses elementos e como é descrita as relações entre dados.

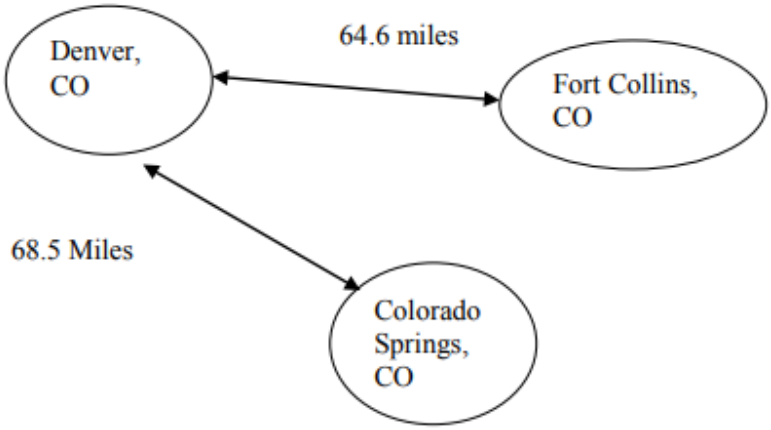

Figura 8 – Exemplo de um banco não relacional retirado da Mason (2015)

2.1.3 Bancos de Dados Relacionais

2.1.3.1 Modelo relacional

O modelo relacional foi proposto em 1970 pelo Ted Codd da IBM Research para representar os dados de uma forma diferente, uma vez que na época eles eram representados de forma hierárquica ou de redes, segundo Elmasri e Navathe (2011). Sua vantagem em relação aos antecessores era a forma simples em que se representava os dados, sendo seu entendimento mais fácil para caso fosse necessário realizar consultas consideradas complexas. Este modelo permitia a descrição de dados de forma natural e simples, por exemplo, por meio da linguagem SQL, além de utilizar o conceito de relação matemática. Com a criação da linguagem SQL os profissionais de sistemas de informação passaram a se preocupar com a especificação das consultas ao invés de se preocupar com os algoritmos para o seu processamento, resultando em ganho de produtividade e sua ampla adoção.

Elmasri e Navathe (2011) explica que Ted Codd montou, junto a uma equipe da IBM, o sistema R. Esse sistema originou o banco de dados relacional que mais tarde evolui para o SQL/DS. A linguagem utilizada pelo sistema da IBM era o Structured Query Language (SQL), sendo essa a utilizada até hoje pela indústria. Com isso, é possível realizar consultas no banco para pegar informações que sejam do nosso interesse, podendo ser apenas uma tabela ou então pegar informações de várias tabelas que se relacionam. Além disso, é possível selecionar quais atributos são desejados para serem retornados de cada tabela consultada, criar ordenações da resposta seguindo um atributo como referência e agrupar informações desejadas. As relações eram definidas por um conjunto de tuplas.

2.1.3.2 Modelo entidade relacionamento

O modelo entidade relacionamento Elmasri e Navathe (2011) é utilizado na parte conceitual de um projeto e tem como intuito ter a descrição dos requisitos de dados, detalhes dos tipos de entidade, relacionamentos e restrições. Ele é formado por entidades que são as tabelas e, além disso, essas possuem atributos que são as características da tabela/entidade. As tabelas podem criar um relacionamento entre tabelas que pode ser de um para um, em que uma instância faz referência a no máximo uma outra instância, um para muitas, no qual que uma instância pode fazer referência a muitas outras instâncias, muitas para muitas, na qual uma instância de uma relação pode ser referência de várias instâncias de outra relação e vice-versa, ou seja, uma instância faz referência a outras instâncias, mas ela pode também ser referenciada por outras da tabela que ela referência.

Graficamente cada uma destas partes são representadas de diferentes formas, sendo elas:

- Retângulo: representa as entidades;
- Losango: representa as relações entre entidades
- Linhas: liga atributos às entidades, ou as entidades com os relacionamentos;
- Elipse: representa os atributos da entidade
- A construção de um arquivo JSON com dados para aplicar os sistemas

Na Figura 9, há um exemplo de um modelo entidade relacionamento.

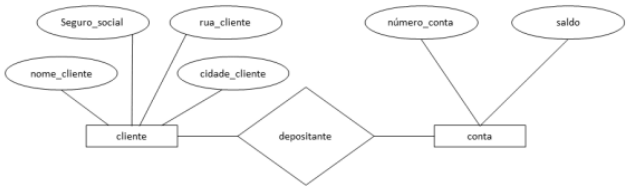

Figura 9 – Diagrama Entidade-Relacionamento. Imagem retirada do livro de Silberschatz (1999)

Atualmente, mesmo com toda a evolução nesta área, o banco de dados relacional continua sendo muito utilizado por oferecer a recuperação de falhas, integridade, rapidez em consultas, segurança, e outras coisas. Isso é possível porque a forma que é realizado o processamento das transações são com base nas propriedades ACID, que já foram definidas anteriormente.

Entretanto, os principais problemas encontrados com a utilização do modelo relacional, de acordo com Fowler e Sadalage (2013), é em relação a sua dificuldade em conseguir

ter escalabilidade. Além disso, realizar mudanças novas de acordo com as mudanças de cenário na empresa é muito custoso, diferente do banco NoSQL. A mesma coisa é dita por Scardoelli e Pinto (2020), ele afirma que isso foi uma das motivações para a criação do banco NoSql.

2.1.4 Armazéns de Dados (Data Warehouse)

Elmasri e Navathe (2011) cita que William H. Inmon define o armazém de dados como uma coleção de dados orientada a assunto, no qual é integrada, não volátil, para ajudar nas decisões das empresas. Com as *data warehouses*, é possível realizar análises complexas, e auxiliar nas tomadas de decisões, a partir dos dados fornecidos. Elas são otimizadas para conseguir recuperar dados e não para operações de transação rotineiras. Essas duas últimas características é o que diferencia elas de um banco de dados. Entretanto, Elmasri e Navathe (2011) como elas têm sido desenvolvidas para atender necessidades específicas, não existe ainda uma definição convencionada, sendo ela definida de diversas formas.

Já a definição de *data warehouse* dada por Smith e Rege (2017), é um agrupamento de dados, no qual onde eles foram retirados pode estar em um único lugar ou separados, são reunidos de forma estratégica que pode ser variada. Os recursos que todo SGBD de data warehouse deve possuir e oferecer, segundo Smith e Rege (2017), deve ser:

- o gerenciamento de grandes volumes de dados;
- carregamento contínuo de dados;
- tipos de dados diferentes dos estruturados;
- consultas repetitivas e análises avançadas;
- consultas em muitos tipos e fontes de dados;
- consultados operacionais de BI;
- alta disponibilidade do sistema;
- níveis de permissão (cientista de dados, minerador de dados, analista de negócios e usuário casual) e área para administração/gerenciamento;

Além disso, segundo Vaisman e Zimányi (2014), por causa do aumento da competitividade no mundo e a rápida mudança que estava ocorrendo, era preciso que elas conseguissem fazer análises complexas para tomar decisões do que seria feito. Os tradicionais bancos de dados não satisfazem essa necessidade, uma vez que eles tinham sido desenhados e otimizados para suportar operações diárias do negócio, e o que precisavam

era de acesso simultâneo de vários usuários, mas, ao mesmo tempo garantir a recuperação caso fosse necessário para ter a consistência dos mesmos. Os bancos de dados operacionais não suportam o histórico de dados e têm um desempenho ruim quando necessitam executar buscas complexas que envolvem muitas tabelas ou carregam grande volume de dados. Desta forma, as *datas warehouses* foram propostas para solucionar a crescente demanda por decisões nas empresas.

Os modelos de *data warehouse* podem ser categorizados em 4 tipos principais, sendo eles:

- tradicional: em que seu principal foco é ter uma coleção de dados estruturados contendo o histórico dos dados. Além disso, ele tem como foco oferecer suporte para relatórios e *dashboards*. Uma das suas principais prioridades é a disponibilização e administração do sistema para realizar consultas.
- operacional: uma *data warehouse* operacional utiliza dados estruturados de forma parecida que o tradicional, porém seu foco principal é o carregamento contínuo de dados para oferecer suporte para aplicativos com análises incorporadas e guardar dados operacionais. Ele deve possuir alta disponibilidade e recuperação dos dados.
- lógico: seu principal objetivo é o armazenamento de grande quantidade de dados e uma variedade grande dos mesmo, podendo armazenar dados de máquina, texto, imagem e vídeo. Por causa desta diversidade, pode ser gerado grandes quantidades de dados, desta forma o gerenciamento destes grandes volumes é crítico. Além disso, ele pode utilizar outras fontes além do DBMS utilizado pela *data warehouse*.
- independente de contexto: este é o único em que se pode declarar novos valores de dados e novos relacionamentos. Ele oferece funcionalidades avançadas como pesquisa e gráficos para ajudar na pesquisa de novos modelos de informação. Além disso, ele consegue realizar consultas em formato livre para oferecer suporte a conceitos muito utilizados na ciência de dados, por exemplo, fazer previsão, mineração de dados e consultas de outras fontes. Por causa da complexidade que as consultas livres podem ter, os variados tipos de dados, a dependência destas consultas e o baixo requisito operacional, ele é mais utilizado pelos cientistas de dados e usuários avançados.

Apesar disso tudo, antes da década de 90 não havia muitos estudos sobre como os armazéns de dados (*Data Warehouses*) seriam utilizados Smith e Rege (2017). Além disso, para conseguir medir o tempo de projeto para criar uma *data wahrehouse*, foi utilizado uma metodologia de gerenciamento de projetos em cascata, no qual utiliza-se um cronograma medido em anos. Devido ao grande investimento que se fazia necessário para desenvolver essa nova tecnologia e os poucos resultados visíveis de funcionamento,

muitas equipes de gerenciamento encerraram estes projetos antes mesmo de entregar algo para usuários finais.

Na década de 90, um grupo de fornecedores de BI, criou aplicativos fáceis de serem utilizados, o que permitia um desenvolvimento mais rápido, além de conseguir fazer análises de relatórios de forma mais eficiente. Eles permitiram ao usuário dividir dados com ou sem uma *Data Warehouse* totalmente desenvolvida. Os fornecedores de DBMS começaram a colocar funcionalidade de BI em seus bancos de dados com tecnologias como o OLAP.

Conforme Smith e Rege (2017), com os novos fornecedores de DBMS, conceitos de modelagem de dados, como o armazenamento de dados operacionais, sendo dados das empresas, dados lógicos. Esses conceitos integrados com os avanços da ferramenta ETL, permitiu maior eficiência na transferência de dados e processamento dos mesmos.

2.2 Ferramentas ETL

Estas ferramentas ganharam popularidade nos anos 70 quando as empresas tinham múltiplos bancos de dados para armazenar diferentes tipos de dados. Como na atualidade o mundo é cada vez mais digital, a troca de informações é muito intensa, estima-se que sejam gerados 2.5 exabytes de dados por dia no mundo segundo McAfee e Brynjolfsson em 2012. Em 2016, segundo Imane e Youness (2017), o Google recebeu 40.000 queries por segundo, o YouTube teve mais de 4 bilhões de visualizações por dia, 700.000 publicações no Facebook por minuto e 200 milhões de tweets postados por dia no Twitter. Desta forma, surgiu o termo *Big Data*, pois as ferramentas tradicionais de análise e processamento de dados não conseguem realizar seu trabalho com essa grande quantidade de dados. Ainda de acordo com o artigo, os dados massivos são caracterizados por 4 componentes, sendo eles:

- volume: significa a quantidade de dados guardada e gerada;
- variedade: tipos de dados podendo ser estruturado, não estruturado, semi-estruturado, como fotos
- veracidade: as informações devem ter qualidade e confiabilidade;
- velocidade: é indicado a velocidade de processamento, em que esta deve ser rápida uma vez que o volume está crescendo.

Com o surgimento de data warehouses Ali e Wrembel (2017), várias fontes de dados heterogêneos passaram a ser integrados e são distribuídos com o objetivo de fornecer acesso unificado e centralizado aos dados para quem for tomar as decisões. Como os dados

originários dos bancos de dados talvez possuam diferentes formatos e modelos, o que pode não ser igual ao da data warehouse, além disso, pode existir inconsistência, redundância ou erros. Devido a tudo isso citado anteriormente, uma ferramenta ETL, que faz extração, transformação e carregamento, pode ser criada, ficando entre as fontes de dados e a *data warehouse*. Assim, o fluxo do ETL inclui a extração e filtro de dados das fontes de dados, transformação dos dados em um modelo em comum, limpeza dos dados para remover erros e valores nulos, padronizar os valores, integrar os dados já limpos em um conjunto de dados, remover valores duplicados e carregá-los para uma *data warehouse*. Ainda será apresentado cada uma das etapas e será mostrado onde cada parte do fluxo é executada. Na Figura 10 , retirada do artigo Ali e Wrembel (2017), é possível ver um exemplo deste fluxo por completo, em que o S1 é a junção de PS1 e PS2 que são duas fonte de dados e S2 é uma outra fonte de dados, os Ann são transformações que precisaram ser feitas nos dados de cada um dos dados e no final é feita uma junção do S1 com S2, sendo feita mais uma transformação e no final carregado para a *data warehouse* representada por DWH.

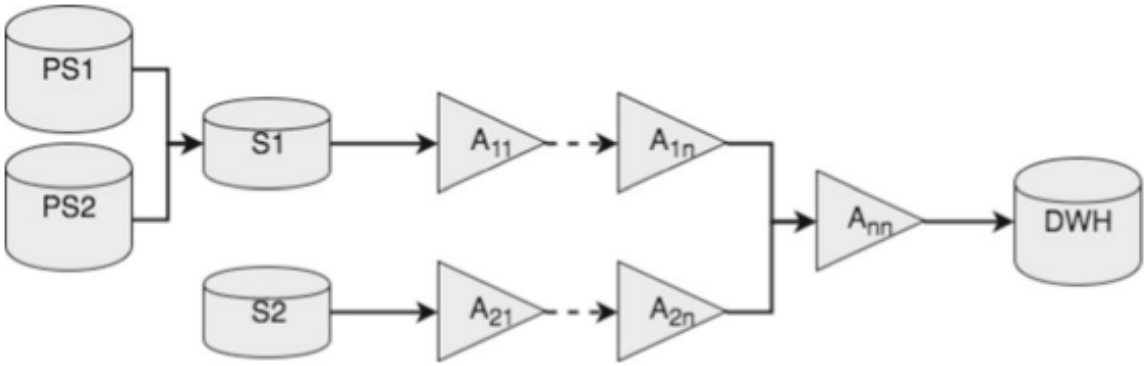

Figura 10 – Exemplo de um fluxo da ferramenta ETL para construir uma data warehouse. Fonte: Ali e Wrembel (2017)

Quando é criada a *data warehouse*, de acordo com o Stonebraker, Ilyas et al. (2018), pode-se assumir que cerca de 15% de todos os dados extraídos estão faltando ou errados em um repositório de uma empresa, é possível utlizar aprendizado de máquina, entretanto é necessário uma grande quantidade de dados que fossem corretos e um modelo de rede neural profunda, só assim seria possível saber as relações complexas entre as entidades. Ainda neste artigo, ele nos informa que na pesquisa dele com cientista de dados, 80% do tempo gasto por eles é procurando bancos de dados para resolver o problema e integração entre eles. Por causa disso, essas ferramentas se tornaram muito importantes.

A definição de cada uma das etapas do ETL, de acordo com o Mali e Bojewar (2015), pode ser dada como:

- Extração: pesquisar e escolher de diferentes bancos de dados quais dados serão

necessários para resolver o problema que se deseja;

- Transformação: a transformação dos dados envolve:
  - Aaplicação de novos regras de negócio;
  - Alimpeza de dados, um dos exemplos dados no artigo é o mapeamento de dados nulos para 0, ou troca "Masculino" para "M" e "Feminino" para "F";
  - Ofiltro de dados, é selecionar apenas as colunas dos bancos que fazem sentido para a solução que está sendo criada;
  - Separar uma coluna em várias, ou fazer o contrário, juntar várias em uma;
  - Juntar vários dados de diferentes bancos de dados;
  - Aplicar uma verificação simples ou complexa para verificar e validar os dados, o exemplo dado pelo artigo é que se as 4 primeiras colunas estiverem vazias retira-se essa linha do processamento;
- Carregamento: esta etapa é responsável por carregar os dados para uma *data warehouse* ou um repositório de dados.

3 Trabalhos correlatos

Segundo Schreiner, Duarte e Mello (2020), a quantidade massiva de dados que são gerados por segundo, trouxe muitos desafios para as aplicações centrada em dados, como, por exemplo, ter que lidar com dados heterogêneos, grande de volume de dados e o rápido crescimento da quantidade de dados gerados. Além disso, muitos desses dados não possuem um esquema já que são estruturados de forma flexível e podem ser heterogêneos. Devido a isso, os bancos NoSQL foram propostos para conseguir solucionar esses problemas que os bancos de dados relacionais, que até então eram muito utilizados, não conseguiam resolver. Os bancos NoSQL têm a capacidade de oferecer alta escalabilidade, armazenamento de dados sem esquema e alta disponibilidade. Alguns exemplos apresentados no seu artigo possuem características de redes sociais, redes de sensores e assistência médica. Schreiner, Duarte e Mello (2020) em seu artigo apresenta um tradutor de esquemas relacionais e instruções SQL para esquemas equivalentes e pode usar métodos dos bancos de dados NoSQL orientado à chave. A abordagem utilizada foi a canônica, chamada *SQLToKeyNoSQL*.

No modelo proposto, um modelo canônico faz o mapeamento de um subconjunto de instruções SQL para uma estrutura hirárquica organizada em uma árvore, que a partir disso pode ser mapeada para qualquer banco de dados NoSQL baseado em chave. Este modelo de banco é o mais simples dos NoSQL, sendo composto por pares de valores chave, é possível pegar os valores através da chave. Neste modelo proposto, mapeia-se um esquema relacional para um modelo canônico intermediário que pega os modelos de dados NoSQL de destino. No modelo canônico chaves e valores são representados em uma estrutura hierárquica simples que é capaz de representar um esquema relacional. Cada chave e valor é representada por um nó. O Schreiner, Duarte e Mello (2020) decidiu por limitar o número de níveis de nós em 3 mais o nó raiz. Devido a isso, ele decidiu por não utilizar outros modelos hierárquicos, como o XML2 e Dom3, pois apesar de serem menos restritos, são mais complexos em tempo.

A arquitetura do *SQLtoKeyNoSQL* é composta por 7 módulos. O primeiro módulo é o *Access Interface*, ele recebe instruções SQL ou uma consulta *Ad-Hoc* e os envia para o módulo que realiza a análise deles. Além disso, ele recebe também os resultados do módulo *Execution Engine* enviando-os para um componente externo.

Schreiner, Duarte e Mello (2020) explica que o módulo *SQL Parser* recebe uma instrução SQL, realiza uma análise sintática e semântica com um dicionário dando suporte. Caso a instrução recebida seja um *SELECT*, *DELETE* ou *UPDATE* ele envia para o módulo de *Query Planner*, nele é realizado uma otimização de *queries*, podendo otimi-

zar os filtros que são conectados pelo operador *AND*. Caso contrário, as instruções são enviadas para o módulo de tradução, chamado *Translator*. Na Figura 11 ele nos mostra do lado esquerdo um exemplo de um arquivo de entrada, já do lado direito está representado a saída dada pelo algoritmo da mesma query de entrada.

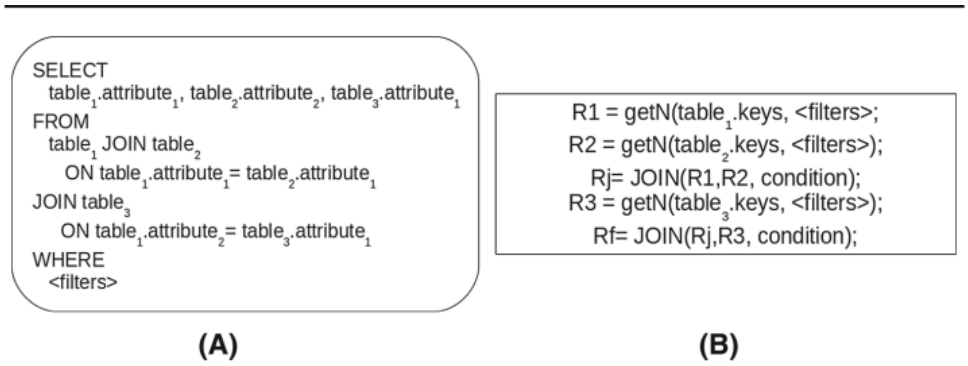

Figura 11 – Exemplo da entrada de um arquivo e saida após a execução do algoritmo.

O módulo *Translator* recebe um ou mais instruções SQL e as traduzem para o formato canônico e é mandado para o módulo de execução, nele é executada as queries com a ajuda do módulo de comunicação e dados armazenados no dicionário. Ele é responsável por processar filtros sobre dados retornados, enviar e receber conjuntos de dados para o módulo *Join Processing*, gerando assim, um conjunto de resultados para serem enviadas para o módulo de interface de acesso.

O módulo *Join Processing* executa as junções entre os diversos conjuntos de dados. Cada uma das operações GetN retorna um conjunto de dados, sendo eles de acordo com os filtros passados como parâmetro. Múltiplas junções são suportadas e são executadas da esquerda para a direita. E por último, tem-se o módulo de comunicação, ele executa requisições de métodos de acesso um ou mais bancos de dados NoSQL. O dicionário, segundo Schreiner, Duarte e Mello (2020), tem como objetivo manter os metadados para cada um dos esquemas do banco relacional, como os atributos, tabelas, chaves primárias, chaves estrangeiras e outros, e informações sobre o banco NoSQL de destino. Sua definição é dada por tuplas, em que o primeiro campo é um conjunto de metadados do banco relacional e o segundo campo é um conjunto de banco de dados NoSQL de destino. A tabela de metadados é formada por uma tupla que contém, nome, ATT, PK, FK, chaves, database em sua formação, onde nome é referente ao nome da tabela, ATT é um conjunto de atributos com o nome das tabelas, PK é a chave primária, FK conjunto das chaves estrangeiras e qual tabela faz referência, podendo ele ser vazia, chaves são o conjunto de chaves, sendo o nome dos nós de terceiro nível no esquema canônico, da tabela, e *database* é o banco de dados NoSQL de destino. Como é possível observar, todo este mapeamento pode ser representado como uma árvore, sendo fácil o entendimento e visualização da hierarquia e relação entre as tabelas. Não serão dados todas as definições trazidas no

artigo, apenas as principais para poder compreender a forma em que foi feito o algoritmo.

O Aftab et al. (2020) aborda sobre a transformação de dados SQL e NoSQL. A forma em que foi proposto é a de um sistema no qual ele identifica os esquemas de dados dinamicamente, e desta forma é realizada a extração, transformação e o carregamento dos dados do NoSQL para o banco de dados relacional. O fluxo seguido, segundo ele, contém os seguintes passos:

- O primeiro é a solicitação de uma novo trabalho de conversão NoSQL para SQL para o ETL, ele irá chamar o analisador de dados integrado para identificar o esquema a partir dos dados do banco NoSQL;
- Encaminha-se um arquivo em formato JSON para realizar a análise dos dados para *SQL query* de acordo com o banco destinado;
- Após a criação do esquema, o ETL processa de forma paralela os dados, que em seguida extrai os dados para processar as *querys* e carregá-las para o banco de dados de destino;

Os tradicionais bancos de dados relacionais são muito utilizados devido a sua eficiência em gerenciar dados, uma vez que ele respeita as propriedades de atomicidade, consistência, isolamento e durabilidade. Entretanto, por causa dessas propriedades, ele não consegue fornecer alta escalabilidade. Por isso, o uso de bancos NoSQL tem crescido muito, junto com o uso de aplicações de larga escala. A flexibilidade em lidar com os dados permite o banco de dados não relacional manipular dados heterogêneos, podendo ter cada instância uma estrutura.

Para conseguir extrair dados de um banco de dados, transformá-los no formato de um outro banco, e carregá-los para o banco de destino, utiliza-se as ferramentas ETL.

Aftab et al. (2020) propõe um sistema que identifica esquema dinamicamente, e então extrai, transforma e carrega os dados de um banco não relacional para um relacional. O sistema criado por ele, segue as seguintes etapas:

- O *Job Manager* é solicitado para realizar a extração, transformação e carregamento de dados NoSQL para SQL. Ele chama o analisador de esquema para conseguir identificar o esquema do banco de dados NoSQL passado. A resposta dada pelo analisador é um esquema em formato *JavaScript Object Notation* (JSON);
- O arquivo JSON com o esquema é analisado e convertido para *queries* seguindo o formato do banco de dados SQL destino;
- Depois da criação do esquema do banco de dados SQL, a ferramenta ETL começa a realizar o processo dados em paralelo partindo do NoSQL para SQL;

- O ETL extrair dados do banco de dados em partes. Após os dados serem extraídos, é começado o processamento dos mesmos para criar consultas no formato do banco de dados de destino que em seguida é populado.

Aftab et al. (2020) explica que o analisador de esquemas foi uma tarefa difícil. Por isso, ele utilizou uma ferramenta de código livre chamada *Variety*, sendo ela um analisador de esquemas para MongoDB. Ele foi desenvolvido para conseguir extrair o esquema de uma coleção de dados. Para conseguir incluí-lo no sistema proposto, foi automatizado para percorrer cada coleção no banco de dados NoSQL e produzir um esquema em formato JSON. O MongoDB é um banco orientado a documentos, em que, é guardado arquivos em JSON sem qualquer restrição de tipos de dados, e alta flexibilidade. Além disso, Aftab et al. (2020) informa que os tipos de chaves primárias não são restritas no *Variety*. Um exemplo dado é que uma chave por ser do tipo inteiro em um documento JSON e outro pode conter o valor como uma *string*, isto dentro de uma mesma coleção. Desta forma, é possível ter o mesmo nome, mas tipos de dados diferentes. Este tipo de heterogeneidade requer um tratamento.

O analisador de esquema examina cada coleção e identifica quais esquemas a serem criados no banco de dados relacional. Além disso, ele percorre os registros do esquema e vai guardando todas as chaves e os valores. Caso essa chave venha a ter um valor do tipo documento JSON, ou matriz JSON, é analisado cada um deles separadamente até não existir nenhum. Para saber quais tipos de dados são contidos no *array*, foi criada uma função que recebe o valor, e retorna verdadeiro caso seja um *array* de JSON, ou falso para qualquer outro caso. Além disso, existe a função que verifica se o objeto é do tipo objeto JSON ou falso para qualquer outro caso. Existe também, as funções de análise do *array* de JSON e de análise de documento JSON, sendo essas funções recursivas. Após isso, os processos ETL são iniciados começando pela transformação do esquema recebido. Cada um destes processos recebem uma parte lógica da informação e do destino. O processo usa a informação da conexão do MongoDB e do nome da coleção para conseguir identificar o esquema.

Para conseguir realizar a criação do banco de dados, as queries são passadas para o destino e executadas. Quando se chega em um dado do tipo JSON, é executada a transformação, criação da tabela e depois é passado o id. Todos os tipos de dados JSON ou *array*, possuem chaves auto incrementadas como inteiro.

Já Iqbal e Daudpota (2006) propõe um framework que utiliza comandos SQL para extrair dados dos bancos de dados relacionais e o converte para um arquivo XML. Após isso, o arquivo XML gerado passa por um módulo, no qual ele deu o nome de *Cleasing Engine*, nele é usado modelos XML predefinidos de acordo com regras de negócios, ele padroniza e limpa os dados. Para finalizar, os dados já padronizados e limpos são mapeados

para o repositório da Warehouse por meio do Mapping Engine.

Segundo explicado por Iqbal e Daudpota (2006) ainda, o componente de extração possui muitas subunidades assim como muitos sistemas de processamento de transações existem em um ambiente de uma data warehouse. Cada subunidade é personalizada de uma forma, extrai dados de seu respectivo sistema e converte os dados para o formato XML de forma genérica. A plataforma de conversão recebe dados para realizar a limpeza de dados a partir da extração utilizando comandos SQL e convertendo-os para documentos xml, juntamente com a especificação do tipo de operadores de limpeza de dados são requeridos. As funções são desenhadas já prevendo que podem conter erros, como por exemplo de tipo, em que um dado pode estar faltando, códigos errados, informações erradas, informações duplicadas e às vezes desnecessárias. Devido a isso, a biblioteca contém funções baseadas em métodos preditivos.

No artigo de Walek e Klimes (2012), é apresentado um algoritmo para converter dados de um banco de dados orientado a documento, uma vez que existem poucos e estes não permitem a migração de chaves estrangeiras. Para a conversão deste banco, o artigo propõe que primeiro seja feita a seleção de arquivos XML do banco de dados orientado a documento que se deseja fazer a conversão para um banco de dados relacional. Além disso, é preciso identificar as dependências entre os arquivos e como se relacionam. Cada um dos documentos representa uma entidade e os atributos são representados em um tipo de dado especificado. Já as relações entre entidades são representadas por campos que possuem um mesmo valor e estão vinculados. Por fim, é necessária a indicação de uma tabela que não possui dependência. Além disso, é feito um mapeamento que terá como resultado uma tupla com 4 elementos, sendo o nome da entidade subordinada a outra, o nome do campo que contém o valor que representa a relação, o nome do tipo de documento que representa a entidade pai e o nome do campo que contém o valor do campo na entidade pai.

4 Ferramenta ETL para conversão de JSON para o modelo relacional

Este capítulo tem como intuito a apresentação das tecnologias utilizadas para o desenvolvimento do projeto, bem como foi desenvolvido, quais funções foram criadas, como foi feito o mapeamento. Além disso, também são apresentados os testes realizados para validar os resultados e se os mesmos estão corretos.

O desenvolvimento do projeto foi realizado por meio da linguagem *javascript* no mecanismo *node.js*, que permite a criação de um serviço com funções que recebem chamadas *http request* que realizam o processamento dos arquivos fornecidos. Após a leitura e processamento dos dados, o serviço salva localmente os resultados em arquivos *JSON*. A linguagem *javascript* foi escolhida devido à facilidade em lidar com os tipos de arquivos JSON, uma vez que a mesma tem funções apropriadas para percorrer um objeto e manipulá-lo, por meio de um *parser* implementado especificamente para o tipo.

4.1 Tecnologias utilizadas

A seguir são apresentadas as tecnologias empregadas no desenvolvimento do protótipo disponibilizado em Lopes (2023):

- *Javascript* foi a linguagem utilizada. Ela é uma linguagem de programação multiplataforma e orientada a objetos para criação de páginas WEB interativas, criada há mais de 25 anos e que conta com mais de 1 milhão de bibliotecas de funções. Os programas criados na linguagem podem ser executados no cliente (navegador web) ou no servidor (por exemplo *Node.js*);
- *Node.js*: foi desenvolvido para criar aplicações com conexões escaláveis e é um ambiente de execução em uma máquina virtual para interpretar e executar os scripts;
- *Mongoose*: é uma biblioteca para o Javascript criar conexão entre um banco MongoDB e o Node.js. Com ela é possível criar modelos de dados, fazer consultas no banco e cadastrar ou remover, ou atualizar dados;
- *Multer*: é um middleware que permite realizar o upload de arquivos e salvá-los localmente após a chamada da API no Node.js e no banco de dados.

4.2 Desenvolvimento

O primeiro passo para conseguir fazer a conversão de um banco NoSQL para SQL é possuir os dados em um ou mais arquivos JSON, para isso, foi criada uma função para guardar os dados localmente para utilizá-los. Na Figura 12 há uma demonstração de como é feita a requisição utilizando a plataforma *insomnia*, nela é possível testar os pontos de acesso do servidor passando as rotas de servidor web, as informações que são necessárias e o tipo de requisição. Ao executar a função, é obtida a resposta:

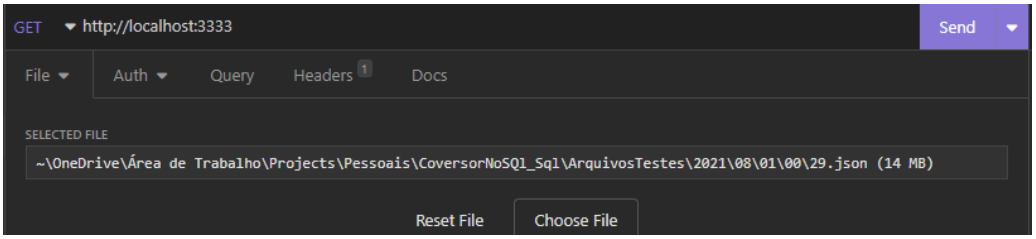

Figura 12 – Upload de arquivo.

Do lado esquerdo é feita a importação dos arquivos, o botão *send* irá executar a função para salvar o arquivo selecionado. Pode-se realizar este procedimento várias vezes para salvar vários arquivos. Esse salvamento ocorre localmente e é possível realizar mudanças mesmo após a execução da função diretamente no arquivo. Esses arquivos podem ser excluídos, ou então, novos podem ser selecionados para serem transformados, mesmo após a conversão sendo feita uma vez, pois estes arquivos não são modificados, apenas lidos para ser feita a análise. A seguir será explicado sobre como foi feita essa análise, quais dificuldades foram encontradas e como foi resolvido. Além disso, na Seção 4.3 é apresentado um exemplo da saída das *querys* para criar e popular cada uma das tabelas do banco, e como é feito para executá-las.

Para realizar os testes deste trabalho, foi utilizado uma base de dados pública com *tweets* feitos na rede social Twitter. Estes arquivos estão no site archive.org (2021). No arquivo usado como teste, os dados contidos possuem informações como o texto publicado, dados do usuário, se ele é verificado na rede social, a descrição de sua página, o nome de usuário, sendo este único, se sua localização está ativada e algumas configurações de seu perfil.

Na Figura 13, é mostrada uma parte do arquivo utilizado como teste e como ele está organizado, tendo o nome da coluna seguido pelo valor, e na Figura 14 é possível visualizar um fragmento formatado do JSON. É possível observar na Figura 13 que cada linha representa uma instância das informações guardadas no banco em que foi retirado junto com todas as suas relações com outras tabelas. Como a quantidade de informações é muito grande em cada uma das instâncias, não é possível visualizar todos os dados delas de uma só vez, pois extrapola a quantidade de caracteres visíveis por linha.

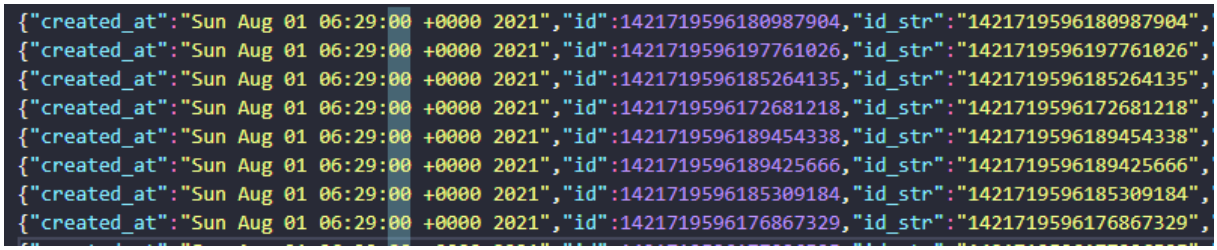

Figura 13 – Exemplo de um fragmento do arquivo utilizado como teste.

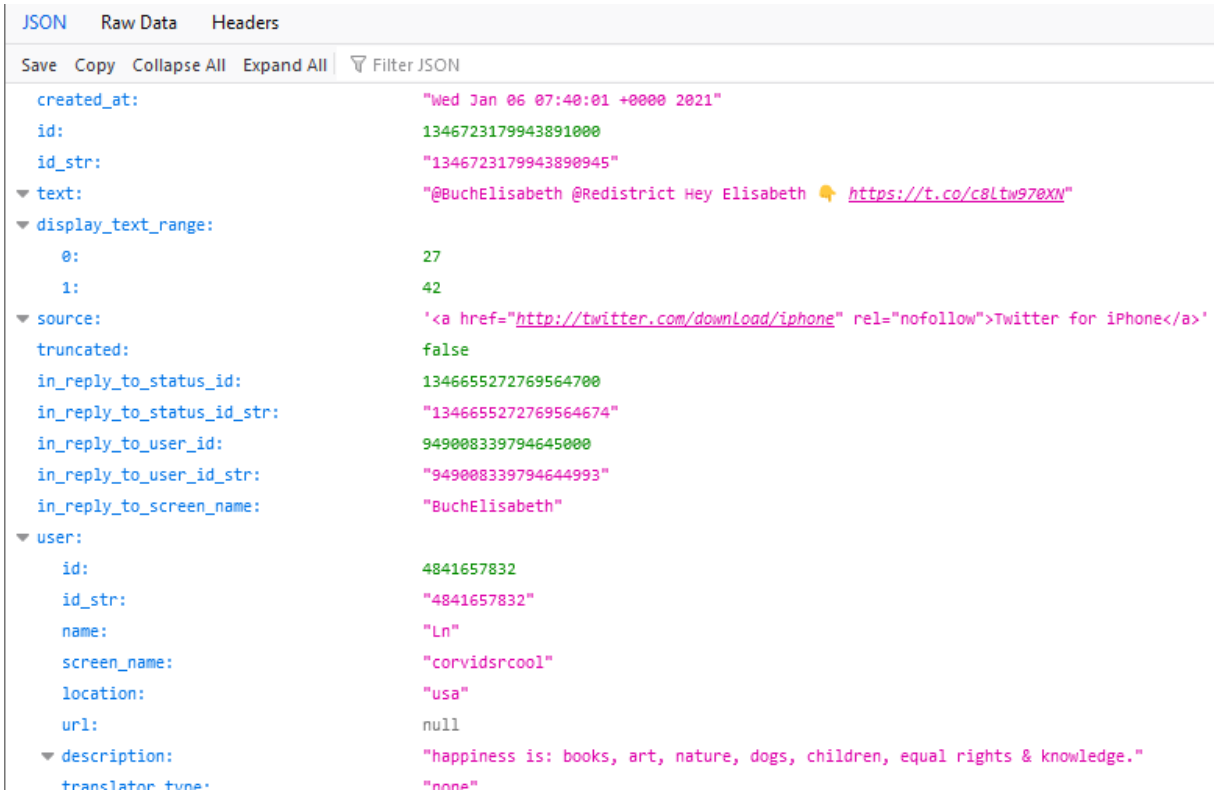

Figura 14 – Visualização de um fragmento do arquivo utilizado como teste.

Comparando com as etapas da ferramenta ETL, esta é a parte da extração, pois é nesta etapa que são reunidos os arquivos de interesse para solucionar o problema. É necessário realizar a busca em bancos de dados, arquivos, sites com informações que fazem sentido e podem complementar aquilo que já se possui. Ela é uma parte importante, pois caso se obtenha dados que não se conectam ou não fazem sentido com o que se deseja criar, então será obtida uma base de dados muito poluída, no final, não faz sentido, ou vai ter muita dificuldade na hora de realizar as transformações necessárias. Entretanto, os dados podem ser juntados com outros para chegar a um resultado desejado, e assim ter uma análise final de melhor qualidade.

Após realizar o salvamento de todos os arquivos, é possível realizar a extração e depois utilizar para guardar suas informações. Tem-se a função que percorre cada um destes arquivos e neles obtêm-se as informações que cada tabela possui, como os seus

atributos e relacionamentos com outras tabelas. Ao ler uma linha de um arquivo, percorre-se ela para fazer a leitura das informações e nome dos campos de cada tabela. Para conseguir saber quais tabelas já foram criadas, existe uma variável que faz este controle, ao começar o mapeamento dos arquivos ela se encontra vazia, mas como não é sabido o nome da primeira tabela, é necessário:

- Dar o mesmo nome do arquivo para esta primeira tabela caso fosse um único arquivo;
- O usuário informar o nome da tabela;
- Dar o nome de tabela pai.

A primeira opção existente é a de verificar se existe mais de um arquivo cadastrado. Essa situação seria muito difícil de se ocorrer, devido a isso ela não foi escolhida. Já a segunda e a terceira são viáveis, mas a a última poderia ser dado o nome muito genérico, que quando fosse fazer análise de dados no banco pronto, poderia perder um pouco o sentido para quem está analisando. Devido a isso, a melhor forma encontrada foi o próprio usuário informar o nome, assim, ele poderá escolher o nome que melhor diz sobre o que aquela primeira tabela informa, e após ele informar é passado para o padrão que foi seguido no projeto, o *snake Case*, o qual substitui os espaços entre palavras por caracteres *underline*.

Após o nome informado, começa-se o mapeamento do arquivo. Primeiramente, pega-se a primeira linha e percorre-se cada chave do objeto, cada uma das chaves serão um atributo da tabela, e é salvo em uma array de arrays, em que cada um dos array possui um objeto, no qual é salvo o nome do atributo juntamente com o seu tipo e a posição em que está. A lista de atributos refere-se a uma tabela que na mesma posição da lista de tabelas, por exemplo, a lista de tabelas sendo [tabela1, tabela2, tabela3] e a array de array com a lista de atributos sendo [[nome: atributo1, tipo: string, nome: atributo2, tipo: number, nome: atributo3, tipo: date], [nome: atributo1, tipo: string, nome: atributo2, tipo: number, nome: atributo3, tipo: date], [nome: atributo, tipo: date]], o primeiro conjunto de atributos refere-se a tabela com nome tabela1, já o segundo refere-se a tabela 2, e assim por diante.

Como já mencionado anteriormente, um problema encontrado é que os arquivos JSON podem possuir tipos de dados não existentes em um banco de dados relacional. Então é necessário realizar uma conversão destes tipos para um que seja válido no novo banco a ser criado. Com isso dito, quando se tem um atributo do tipo array, o esquema sendo analisado não está na primeira forma normal, e para conseguir isso é preciso eliminar este atributo multivalorado. Para isso, eles devem ser convertidos em relacionamentos entre outras tabelas, por exemplo, o tipo de dado objeto, que é encontrado no JSON, pode ser interpretado como um relacionamento entre uma ou mais tabelas, sendo o objeto

uma nova tabela em que a tabela pai faz referência. Desta forma, a chave primária da tabela filha passa para a pai, assim no futuro é possível consultar com qual instância da tabela filha ela está se relacionando. O mesmo deve ser realizado para o tipo de dado que é um array, os dados contidos nela são convertidos em uma nova tabela que vai conter as informações do array e um atributo que vai ser uma chave estrangeira da tabela que faz referência. Quando feito isso, garante-se que o esquema está na primeira forma normal, pois vai existir apenas atributos atômicos.

Um outro problema encontrado foi saber qual chave seria a primária caso houvesse, e se não houvesse, qual atributo seria utilizado como tal. Para conseguir solucioná-lo, foi criado um atributo novo como chave, pois esta forma seria a mais simples de ser feita, uma vez que caso exista um atributo chave na tabela, há duas formas de saber, ou o usuário nos informaria qual atributos seria a chave primária dentre os atributos existentes, mas desta forma seria inviável, pois caso existam muitas tabelas, o usuário teria que digitar todas. Outra forma seria a de percorrer todas as instâncias e ver quais atributos não repetem, e escolher um deles, porém, desta forma poderia ser escolhido um atributo que no futuro poderia vir a ter valores iguais, gerando assim uma inconsistência, além disso, poderia não existir uma chave.

Como há apenas uma chave primária por tabela, pode-se dizer que o esquema está na segunda e terceira forma normal, pois os atributos não primários dependem da chave primária por inteiro, não existindo dependência parcial, o que garante estar na segunda forma normal.

Com o tipo de dado array, é permitido ter dois tipos de relacionamento, sendo um para muitos ou muitos para muitos. Com a separação feita na primeira forma normal pode ocorrer redundância das informações mudando apenas o valor da chave estrangeira quando se possui uma relação muito para muitos. Entretanto, para saber com qual caso é, se faz necessário percorrer todas as instâncias existentes. Caso alguma repetir uma das informações, então é do tipo muitos para muitos, se nenhuma repetir é um para muitos. Caso seja uma relação muitos para muitos, será criada uma nova tabela com a chave estrangeira de cada uma das tabelas, se for a um para muitos, será passada a chave primária da tabela pai para a filha. Desta forma garante-se que está na quarta forma normal.

As informações dos nomes das tabelas são guardadas em um array e os atributos de cada tabela em um outra variável que é uma matriz, em quem a uma linha faz referência a uma posição do array com os nomes das tabelas, por exemplo, é obtido o array com o nome das tabelas, ['tabela1', 'tabela2'], e o array com os atributos, [['atributo1', 'atributo2'], ['atributo3', 'atributo4']], o primeiro conjunto da tabela de atributos faz referência a tabela 1, e o segundo conjunto faz referência a tabela 2 isso também se repete. No array de atributos, obtêm-se vários objetos em que uma das informações guardadas é o nome

do atributo e a outra informação é o seu tipo, que serão os compatíveis com o modelo relacional, por exemplo, string, data, booleano, inteiro, real.

Ao finalizar o percurso de todos os arquivos. Então é passado o resultado para uma outra função que irá montar as *queries* para a criação do banco de dados relacional. Assim, é possível copiar o resultado e executar no SGBD a criação das tabelas com seus atributos, fazendo também a inserção de dados já existentes nos arquivos JSON que foram extraídos. Para conseguir construí-las, são percorridos os arrays com as informações dos atributos da tabela, caso nenhum desses atributos consultados for uma chave estrangeira que se refere a uma tabela que ainda não foi construída, é realizada a construção da *query*. Para isso, pega-se posição que está na lista de atributos e consulta-se o nome da tabela, em seguida percorre-se novamente a lista de atributos e é inserida um a um com seu respectivo tipo de valor a ser recebido.

Como pode ocorrer casos em que tabela que está sendo consultada para criação depende de atributos de outras tabelas, e esta pode não ter sido criada ainda, não pode montar a *query* dela, pois o banco irá retornar um erro na hora de executá-las. Por isso, é necessário criar a tabela que ela depende primeiro e depois criá-la. Devido a isso, é necessário, múltiplas passagens pelo array, até que o número de tabelas criadas seja igual o número de tabelas que estão na variável com os nomes das tabelas. Para não precisar fazer essas múltiplas passagens, poderia ter como saída diferentes arquivos, onde cada um faz referência a uma tabela, mas isso poderia se tornar um problema, pois se a quantidade de tabelas criadas for muito grande, pode se tornar uma tarefa inviável de copiar as *querys* para executar no SGBD. Além disso, pode ocorrer confusão ao não saber qual ordem executar por causa das dependências de tabelas. Desta forma, uma das *quueries* pode ser executada precisando de uma outra tabela, e ela não ter sido criada ainda,será retornado um erro. Entretanto, para conseguir procurar o arquivo certo, poderá demandar um esforço muito grande. Então, a melhor forma encontrada foi de retornar para o usuário apenas um arquivo com todas as *querys* em ordem que devem ser executadas.

Resumindo, as tabelas são mapeadas da seguinte maneira:

- Cria-se uma tabela pai, que conterá todas as informações das tabelas no primeiro nível.
- Se um determinado dado for de algum dos tipos que existem no SQL, então é apenas acrescentado no array de lista de atributos na posição referente a tabela que está sendo lida junto com seu nome e o nome do tipo
- Se um determinado dado for do tipo objeto, então este objeto em uma nova tabela, pega-se a chave desta tabela e salva na tabela pai.
- Se um determinado dado for um array, então é obtido 3 possíveis situações:

  - a primeira é que o tipo de dado nele se difere de objetos, desta forma está sendo lidado com um atributo multivalorado, para isso é criado uma nova tabela com a chave da tabela pai e um valor.
  - Se for uma relação NxM se terá objetos dentro do array, e para descobrir essa relação, uma vez que a relação NxM e 1xN vão ser objetos, é necessário percorrer todos os arquivos para verificar se existe alguma outra instância que repete algum valor, e deve-se fazer isso com todos os possíveis valores, pois a que está sendo analisado no momento no pior caso não terá nenhuma repetição, mas outra instância pode conter valores que aparecem em outra. Desta forma, caso apareça alguma repetição, é reconhecido como uma relação NxM, e desta forma é criado uma nova tabela com a chave primária de cada das tabelas e uma outra para guardar as informações da nova tabela.
  - Se for uma relação 1xM, também serão obtidos objetos no array. Desta forma é feito da mesma forma ao que foi realizado na relação NxM, entretanto, se houver um mapeamento de todas os valores entre eles mesmos, e não for encontrada nenhuma repetição, então será uma relação 1xN, então é criado uma nova tabela e passa-se a chave primária da tabela pai para a tabela filho.

- Finalizado a conversão de todos os arquivos, é passado para a função que monta as querys e em seguida retorna um arquivo contendo como conteúdo as querys para que o usuário possa fazer o download e executá-las no SGBD.

O código pode ser encontrado no github Lopes (2023). A fim de melhor entendimento o pseudocódigo pode ser descrito como:

1. Carrega todos os arquivos salvos localmente;
2. Percorre as informações guardadas;
3. Pega-se a primeira linha do arquivo e percorre ela;
4. Verifica se o tipo de atributo da instância sendo lida no momento, caso seja um objeto é criado um atributo que faz referência a tabela filha, pois esse objeto também é transformado em uma tabela. Caso seja um *array* é executada a função que está no próximo pseudo código .
5. caso o retorno da função que verifica o tipo de relação retorna NxM, cria-se uma nova tabela com os *ids* de cada tabela. Caso ela retorna 1XN então cria-se um atributo na nova tabela que faz referência a tabela do objeto dentro do array, em que este refere-se ao id do pai.

6. Após isso os objetos são percorridos dentro do array para criar os atributos da nova tabela.

7. Se as informações dentro do array se diferirem de um objeto cria-se uma tabela para guardar o valor e o *id* da tabela pai, pois ele é considerado um atributo multivalorado.

8. Isso é feito até acabar a linha, depois vai para a próxima e volta para o passo 3. Caso seja a última, é passado para o próximo arquivo e volta para o passo 2.

Existe uma variável para guardar o nome da tabela, outra que é uma matriz para guardar os atributos junto com o tipo de dado que ela recebe, sendo cada linha referente a uma tabela e uma última variável em que é guardada as informações, em que cada linha possui um conjunto de informações e nessas informações são guardadas os dados de cada linha do arquivo JSON original.

Em seguida, os seguintes passos são executados para verificar o tipo de relação entre as tabelas;

1. Recebe-se o caminho para chegar até o atributo a ser verificado;

2. Percorrem-se todas as linhas de todos os arquivos comparando se a informação atual lida é igual a alguma outra, caso sim, retorna-se a string NxM, caso todas as instâncias sejam percorridas comparando com todas as outras e nenhuma seja igual, retorna-se a string 1xN.

4.3 Testes

Nesta seção serão apresentados os resultados encontrados a partir do projeto criado, como foi a base testada, comparando com o artigo do Aftab et al. (2020), se foi obtido o mesmo resultado ou não. É necessário também ver a eficiência do algoritmo, verificando quanto tempo demora para converter o arquivo JSON para SQL da base de dados do Twitter citada anteriormente.

Primeiramente, foram separados os 3 modelos JSON apresentados no artigo e é necessário executar os códigos. No final é necessário comparar os resultados, se foram iguais ou se houve alguma diferença e o motivo de ter ocorrido. Para isso o primeiro modelo do arquivo json apresentado foi o da Figura 15, nele obtém-se os atributos a esquerda e seus respectivos valores à direita. Na Figura 16, é possível observar o diagrama do artigo com o nome da tabela e os atributos mapeados. Na Figura 17, é possível observar o diagrama do resultado encontrado pelo algoritmo com o nome da tabela e os atributos mapeados. Fazendo uma comparação, o nome da tabela é diferente, pois no algoritmo o

nome é dado pelo usuário, e como estava sendo feito testes, foi dado o nome como teste, mas poderia ser trocado para o mesmo do artigo. O segundo exemplo dado no artigo é mostrado na Figura 18.


```
{
  "amount": "361.46",
  "date": "2012-02-01T19:00:00.000Z",
  "business": "Bogisich - Rogahn",
  "name": "Home Loan Account 4758",
  "type": "deposit",
  "account": "96170278"
}
```


Figura 15 – Exemplo do primeiro arquivo JSON.

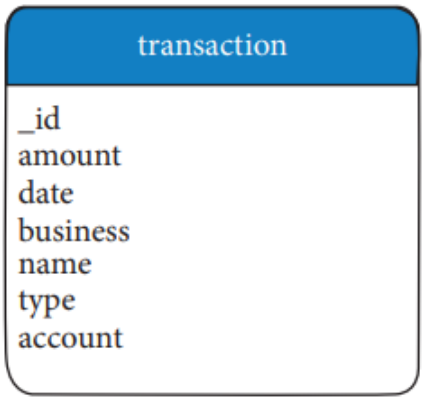

Figura 16 – Print do resultado do exemplo 1 encontrado pelo algoritmo proposto.

Na Figura 19 é possível observar o diagrama do artigo com o nome da tabela e os atributos mapeados.

Na Figura 20 é possível observar o diagrama do resultado encontrado pelo algoritmo com o nome da tabela e os atributos mapeados.

No artigo não foi mostrado como o atributo geo foi mapeado, já no proposto para este trabalho, como no arquivo JSON ele recebe um objeto, é convertido em uma nova tabela, depois passado o *id* para a tabela *address*. Desta forma, é possível saber qual instância da tabela *geo* referencia a instância da tabela address. Além disso, no algoritmo proposto para este trabalho, é possível observar o nome das chaves estrangeiras das tabelas, o que não foi apresentado no artigo, entretanto pode-se ver que elas se relacionam através das linhas entre cada tabela.

Figura 17 – Print do resultado do exemplo 1 encontrado pelo algoritmo proposto.


```
{
    "name": "Fletcher",
    "username": "Fletcher_Hudson",
    "avatar":
"https://s3.amazon.com/faces/twitter/michaelkoper/128.jpg",
    "email": "Fletcher_Hudson42@gmail.com",
    "dob": "1956-12-28T16:25:07.727Z",
    "phone": "492-833-6382 x84651",
    "address": {
          "street": "Antonetta Course",
          "suite": "Apt. 012",
          "city": "Sawaynville",
          "zipcode": "11342-2477",
          "geo": {
             "lat": "–36.5510",
             "lng ": "–24.2751"
          }
    },
    "website": "chasity.net",
    "company": {
          "name": "Flatley Group",
          "catchPhrase": "Customer-focused radical installation",
          "bs": "ubiquitous iterate systems"
       }
}
```


Figura 18 – Exemplo do segundo arquivo JSON.

Na Figura 21 é possível observar que *address* possui um aninhamento de objeto no qual *geo* é um objeto contido nele. Já *phone* é um vetor de *strings* no qual possui o número de telefone da pessoa e *age*, *name* possui um valor sendo o primeiro uma *string* e o segundo um número inteiro. O resultado deste mapeamento, a partir do código criado, pode ser observado na Figura 22, em que *geo* se relaciona com *address* é que este se relaciona com

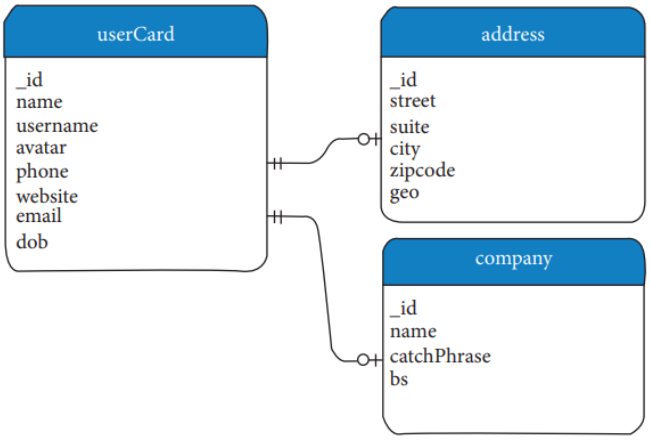

Figura 19 – Print do resultado encontrado pelo algoritmo proposto.

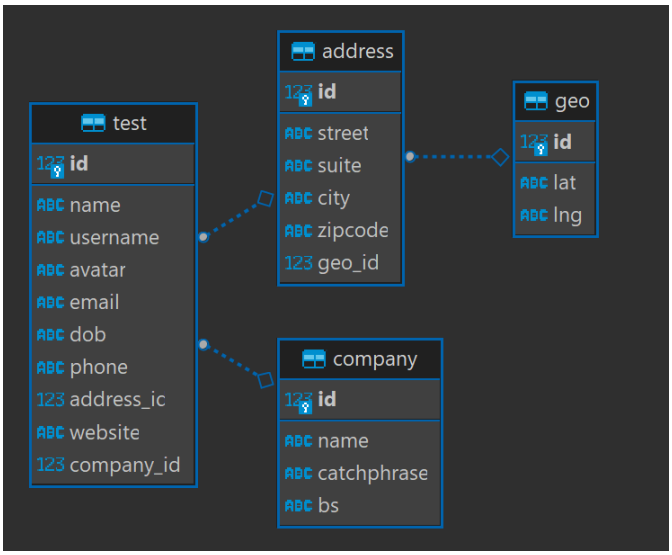

Figura 20 – Print do resultado do exemplo 2 encontrado pelo algoritmo proposto.

test, já a tabela *company* e *phones* se relaciona unicamente com *test*. A tabela *test* possui as chaves primárias de *company* e *address* já que ela se relaciona uma unica instância de cada tabela e *phone* a chave de *test* já que se relaciona uma unica vez com *test*.

Na Figura 23 existem duas instâncias iguais a anterior, diferenciando apenas no atributo *phones* no qual é uma relação NxM, o resultado obtido na execução do algoritmo pode ser observado na Figura 24, e sua única diferença do anterior é a criação de um tabela a mais para guardar o valor da relação *phones test* com as chaves primárias de cada um.

Figura 21 – Print do resultado do exemplo 2 encontrado pelo algoritmo proposto.

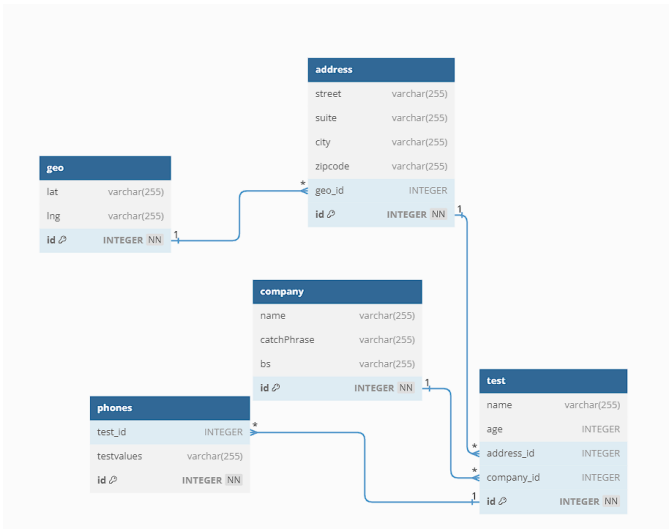

Figura 22 – Print do resultado do exemplo 2 encontrado pelo algoritmo proposto.

```
{
    "name": "Alysa Feil",
    "age": 20,
    "address": {
        "street": "Beatty Lights",
        "suite": "Suite 551",
        "city": "South Hubert",
        "zipcode": "12402",
        "geo": {
            "lat": "88.4538",
            "lng ": "85.3045"
        }
    },
    "phones": ["(355) 681-7737", "(355) 681-7738", "(355) 681-7739" ],
    "company": {
        "name": "Aufderhar and Sons",
        "catchPhrase": "Cross-group multi-state ability",
        "bs": "mission-critical reintermediate e-markets"
    }
}
{
    "name": "Alysa Feil 1",
    "age": 22,
    "address": {
        "street": "Beatty Lights 1",
        "suite": "Suite 5511",
        "city": "South Hubert 1",
        "zipcode": "12402",
        "geo": {
            "lat": "88.4539",
            "lng ": "85.3046"
        }
    },
    "phones": ["(355) 681-7737", "(355) 681-7738", "(355) 681-7740" ],
    "company": {
        "name": "Aufderhar and Sons 1",
        "catchPhrase": "Cross-group multi-state ability 1",
    "bs": "mission-critical reintermediate e-markets 1"
    }
}
```


Figura 23 – Print do arquivo Json utilizado para teste.

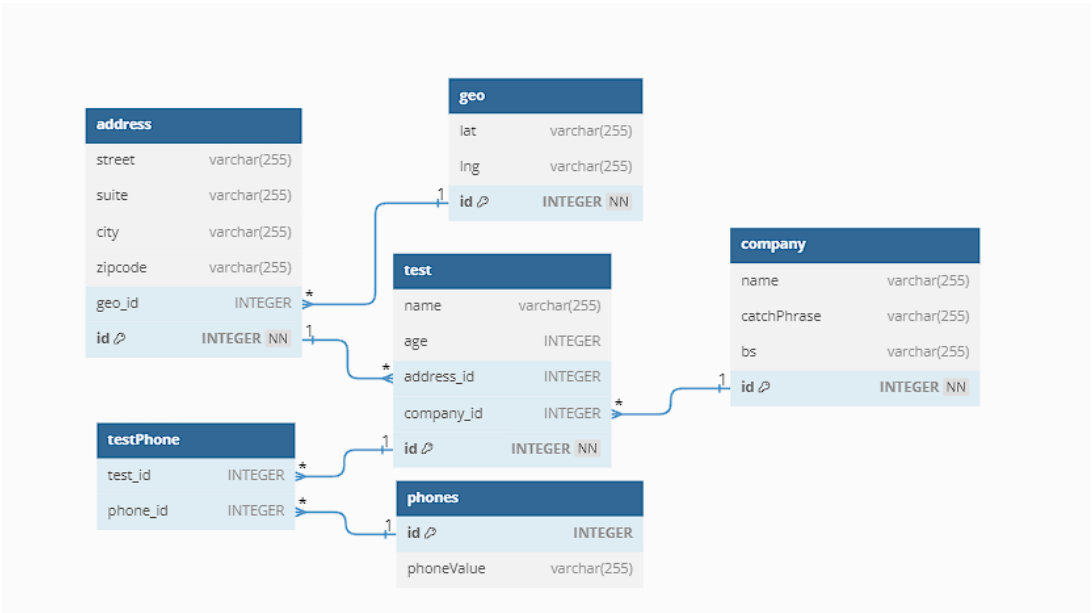

Figura 24 – Print do esquema resultante.

5 Conclusão

Hoje, com o mundo caminhando cada vez mais para ser mais digital, empresas estão a todo momento tentando salvar dados das pessoas que podem beneficiar as estratégias de negócios da empresa. Como cada uma possui seu modelo de negócio, um tipo de banco de dados pode se mostrar mais vantajoso que outro. Entretanto, quando vão pegar dados de um outro banco de dados eles podem ser de tipos incompatíveis, sendo necessário assim, um serviço manual e podendo ser muito demorado e inviável.

Para conseguir realizá-lo, foi necessário o entendimento das diferenças entre cada tipo de banco de dados, quais tipo de atributos um tinha que o outro não, qual atributos de um poderia ser convertido no outro, como deveria ser feita a conversão para este tipo. Além disso, as relações não eram explícitas, ou seja, vinham com todos os objetos juntos, desta forma deveria ser feita uma passagem dos dados para verificar qual tipo de relacionamento entre as tabelas possuía. Também para saber quais atributos possuía uma tabela, era preciso percorrer o objeto por completo e ver o tipo e nome de cada chave do objeto.

Portanto, com esse trabalho foi capaz de ver uma possível forma de fazer a conversão destes tipos de dados para um outro modelo de banco, sendo aqui mostrado do banco de dados não relacional para o relacional. O maior desafio encontrado foi conseguir mapear as relações entre as tabelas, pois existem 4 tipos, sendo dois fáceis de serem identificados. Como era um para muitos ou muitos para muitos, um poderia ser feito apenas o passamento da chave da tabela pai para filha. Entretanto, para achar as outras duas era necessário realizar uma passagem para verificar se existiam repetições de algum objeto, caso houvesse, seria sabido que era uma relação muitos para muitos, caso contrário era uma relação um para muitos. Esta tarefa pode ser muito custosa, pois pode facilmente se tornar algo com tantas iterações, que tornaria inviável esperar para obter o resultado da mesma.

No artigo de Aftab et al. (2020), não foi informado se para avaliar a performance foi utilizado o mesmo modelo relacional que os anteriores, apenas a quantidade de instâncias e o tempo gasto. Desta forma, não será comparado diretamente com o artigo e o algoritmo, mas mostrará o tempo encontrado pelo artigo com a quantidade de instâncias informadas e o tempo gasto por ambos. Para isso, serão utilizados os arquivos da base pública com informações do twitter. Este teste terá o mesmo modelo do artigo, será executado 3 vezes o código e no final será feita uma média do tempo gasto.

No primeiro teste do artigo foram executadas 100 mil instâncias, a primeira execução durou 5.33 segundos, a segunda 5.30 segundos e a terceira 5.55 segundos, tendo

assim uma média de 5.39 segundos. Já no algoritmo, o primeiro teste foi realizado com 5500 instâncias, na primeira execução a duração para conseguir gerar ler todo o arquivo e montar o arquivo com as *querys* foi de 20.13 segundos, a segunda 22.38 segundos já a terceira foi de 21.52 segundos, desta forma é obtida uma média de 21.34 segundos.

Esta grande diferença pode ter ocorrido por 2 fatores:

- um sendo a quantidade de atributos que possui, pois se forem muitos atributos, a quantidade de leitura de cada atributo é grande para percorrer, pois em uma instância pode existir dados que outras não possui;
- Outro problema é a quantidade de tabelas encontradas pode ser diferente. O arquivo utilizado no algoritmo criado, foram mapeadas 37 tabelas. Entretanto, se no exemplo do artigo tiver sido utilizado um arquivo que a quantidade de tabela mapeada for de apenas 4 tabelas, este tempo seria reduzido significativamente.
- Caso as tabelas possuam muitos atributos, e eles seram possíveis de receber mais de uma informação, ou seja, um array, a quantidade de vezes que seria necessário percorrer todas as instâncias para verificar seu tipo de relação, podendo ser do tipo um para muitos ou muitos para muitos, demandaria muito tempo.

Pode ocorrer mais de um destes problemas, prejudicando ainda mais o tempo. Será apresentada uma das possíveis formas de tentar reduzir o tempo, principalmente na questão de verificar o tipo de relação.

5.1 Trabalhos futuros

Para que este trabalho pudesse ser realizado, era necessário ter o conhecimento da matéria de banco de dados, além de conhecimentos de como manipular arquivos JSON.

Em trabalhos futuros poderá ser feita a conversão inversa, indo do relacional para o não relacional. Ela é semelhante, pois o retorno das pesquisas de um banco relacional retorna uma tabela, onde cada linha é uma instância e cada coluna um atributo. Então, é possível fazer de forma semelhante à que foi realizada aqui e podendo aproveitar as tabelas já criadas sem precisar de novas, uma vez que da forma que foi realizada neste trabalho existiam tipos de dados do JSON que não havia no relacional, sendo necessário a criação de tabelas dependendo do tipo.

Além disso, poderá ser feito o estudo para realizar uma otimização do algoritmo de transformação, pois a quantidade de passagens em todas as instâncias de todos os arquivos podem tornar a tarefa extremamente demorada e até inviável. Por exemplo, caso o tipo de relação de um atributo seja muito para muitos, então não se faz necessário mais a passagem de todas as instâncias novamente, uma vez que este já é o pior caso. Além

disso, colocar uma validação a mais em que, caso já tenha sido mapeado aquele atributo, não realizar a validação de novo. Uma vez atribuida, ele procura em todas as instâncias e em todos os arquivos para verificar seu tipo, uma forma de ser feito isso, é colocar quais atributos já foram mapeados, e se ele já estiver na lista passar para o próximo.

Referências

AFTAB, Z.; IQBAL, W.; ALMUSTAFA, K. M.; BUKHARI, F.; ABDULLAH, M. Automatic nosql to relational database transformation with dynamic schema mapping. **Scientific Programming**, Hindawi Limited, Article ID 8813350, p. 1–13, 2020. doi:10.1155/2020/8813350. Citado 6 vezes nas páginas 5, 8, 29, 30, 39 e 45.

ALI, S. M. F.; WREMBEL, R. From conceptual design to performance optimization of etl workflows: current state of research and open problems. **The VLDB Journal**, Springer, v. 26, n. 6, p. 777–801, 2017. doi:10.1007/s00778-017-0477-2. Citado 3 vezes nas páginas 5, 24 e 25.

ARCHIVE.ORG. **Base de dados do twitter**. 2021. Disponível em: <https://archive.org/details/archiveteam-twitter-stream-2021-08/>. Acesso em: 23 mar. 2020. Citado na página 33.

Bernard Marr. **How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read**. 2018. Disponível em: <https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/?sh=46c3c0a60ba9>. Acesso em: 21 mai. 2018. Citado na página 7.

ELMASRI, R.; NAVATHE, S. **Sistemas de banco de dados**. Editora: Pearson, 2011. Citado 13 vezes nas páginas 5, 7, 8, 12, 13, 14, 15, 16, 17, 18, 20, 21 e 22.

FOWLER, M.; SADALAGE, P. J. **NoSQL Essencial: um Guia Conciso Para o Mundo Emergente da Persistência Poliglota**. São Paulo: Novatec Editora, 2013. 216 p. ISBN 978-85-7522-338-3. Citado na página 21.

HARRINGTON, J. L. **Relational database design and implementation**. Editora: Morgan Kaufmann, 2016. Citado na página 11.

IMANE, L.; YOUNESS, T. State of the art in mapreduce: Issues and approaches. In: **Proceedings of the 2nd International Conference on Big Data, Cloud and Applications**. New York, NY, USA: Association for Computing Machinery, 2017. (BDCA'17). ISBN 9781450348522. Disponível em: <https://doi.org/10.1145/3090354.3090397>. Citado na página 24.

IQBAL, T.; DAUDPOTA, N. XML based framework for ETL processes for relational databases. **WSEAS Transactions on Information Science and Applications**, v. 3, n. 7, p. 1402–1406, 2006. http://www.wseas.us/e-library/conferences/2006hangzhou/papers/531-620.pdf, https://dl.acm.org/doi/10.5555/1973598.1973691. Citado 2 vezes nas páginas 30 e 31.

LOPES, V. H. E. **Código fonte do trabalho**. 2023. Disponível em: <https://github.com/victorhelopes/TCC>. Acesso em: 23 mar. 2023. Citado 2 vezes nas páginas 32 e 38.

MALI, N.; BOJEWAR, S. A survey of ETL tools. **International Journal of Computer Techniques**, Oct, v. 2, n. 5, p. 20–27, 2015.

http://www.ijctjournal.org/Volume2/Issue5/IJCT-V2I5P3.pdf. Citado na página 25.

MASON, R. T. Nosql databases and data modeling techniques for a document-oriented nosql database. In: **Proceedings of Informing Science & IT Education Conference (InSITE)**. Denver, CO, USA: Informing Science Institute, 2015. v. 3, n. 4, p. 259–268. https://proceedings.informingscience.org/InSITE2015/InSITE15p259-268Mason1569.pdf. Citado 2 vezes nas páginas 19 e 20.

POKORNY, J. Nosql databases: A step to database scalability in web environment. In: **International Conference on Information Integration and Web-based Applications and Services (iiWAS)**. Ho Chi Minh City, Vietnam: ACM, 2011. (iiWAS '11), p. 278–283. doi:10.1145/2095536.2095583. Citado na página 18.

ROB, P.; CORONEL, C. **Sistemas de Banco de Dados: Projeto, Implementação e Gerenciamento**. Editora: Cengage, 2011. Citado na página 11.

SCARDOELLI, K. A.; PINTO, G. S. Banco de dados nosql: uma alternativa para grandes empresas. **Revista Interface Tecnológica**, v. 17, n. 2, p. 219–230, 2020. doi:10.31510/infa.v17i2.949. Citado na página 22.

SCHREINER, G. A.; DUARTE, D.; MELLO, R. dos S. Bringing SQL databases to key-based nosql databases: a canonical approach. **Computing**, v. 102, n. 1, p. 221–246, 2020. doi:10.1007/s00607-019-00736-1. Citado 3 vezes nas páginas 11, 27 e 28.

SHARMA, V.; DAVE, M. Sql and nosql databases. **International Journal of Advanced Research in Computer Science and Software Engineering**, v. 2, n. 8, 2012. (journal descontinuado, link em archive.org). Citado 2 vezes nas páginas 5 e 19.

SILBERSCHATZ, A. Henry f. korth. **Sudarsham, Database System Concept, McGrsw–Hill**, 1999. Citado na página 21.

SMITH, J.; REGE, M. The data warehousing (r) evolution: Where's it headed next? In: **Proceedings of the International Conference on Compute and Data Analysis**. New York, NY, USA: Association for Computing Machinery, 2017. (ICCDA '17), p. 104–108. ISBN 9781450352413. Disponível em: <https://doi.org/10.1145/3093241.3093268>. Citado 3 vezes nas páginas 22, 23 e 24.

STONEBRAKER, M.; ILYAS, I. F. et al. Data integration: The current status and the way forward. **IEEE Data Eng. Bull.**, v. 41, n. 2, p. 3–9, 2018. Citado na página 25.

VAISMAN, A. A.; ZIMÁNYI, E. **Data Warehouse Systems: Design and Implementation**. Berlin Heidelberg: Springer, 2014. (Data-Centric Systems and Applications). doi:10.1007/978-3-642-54655-6. ISBN 978-3-642-54654-9. Citado na página 22.

VERA, H.; BOAVENTURA, W.; HOLANDA, M.; GUIMARãES, V.; HONDO, F. Data modeling for nosql document-oriented databases. In: **CEUR Workshop Proceedings**. Brasília: University of Brasília, 2015. v. 1478, p. 129–135. Citado 2 vezes nas páginas 5 e 19.

WALEK, B.; KLIMES, C. Data Migration between Document-Oriented and Relational Databases. **International Journal of Computer and Information Engineering, World Academy of Science, Engineering and Technology**, Zenodo, v. 6, n. 9, p. 1144–1148, 2012. doi:10.5281/zenodo.1059451. Citado na página 31.