JTSQL: UMA CLI PARA TRANSPILAÇÃO DE MODELOS DE DADOS JSON PARA INSTRUÇÕES DDL EM SQL

Lucas Baldasso[1], Giovane Galvão[2]

[1] Discente do Centro Universitário Campo Real

[2] Docente do Centro Universitário Campo Real

Rua Comendador Norberto, 1299 - Santa Cruz – Guarapuava – PR – Brasil

{engs-lucasbaldasso@camporeal.edu.br, prof_giovanegalvao@camporeal.edu.br}

Resumo

A modelagem e criação de bancos de dados é uma etapa fundamental no desenvolvimento de softwares. Neste trabalho, foi desenvolvido um protótipo de uma ferramenta de transpilação de modelos de representação de dados *Javascript Object Notation* (JSON) para instruções de *Data Definition Language* (DDL) em *Standard Querying Language* (SQL) com a sintaxe do PostgreSQL com intuito de ser uma ferramenta educativa. A ferramenta foi desenvolvida utilizando C# e .NET e foi distribuída em forma de *Command Line Interface* (CLI) para que possa ser incorporada em qualquer ambiente. A ferramenta conta com um argumento e nove parâmetros que customizam a execução e transpilação da estrutura de dados. A ferramenta atendeu as expectativas, mas demonstrou limitações quanto a personalização de campos individuais. Portanto, uma série de melhorias poderia fazer com que a ferramenta fosse utilizada tanto para fins educativos quanto para o dia a dia de trabalho.

Palavras-chave: Modelagem de dados; Ferramentas de ensino de bancos de dados; Ensino de bancos de dados.

Abstract

Database modeling and creation are fundamental steps in software development but can be challenging for novice developers. This study presents the prototype of a transpilation tool that converts Javascript Object Notation (JSON) data representation models into Data Definition Language (DDL) instructions with the aim of serving as an educational resource. The tool was developed using C# and .NET and is distributed as a *Command Line Interface* (CLI) application to allow integration into any environment. It includes one main argument and nine parameters that customize the execution and data structure transpilation. While the tool met the expected objectives, it demonstrated limitations in customizing individual fields. Thus, a series of improvements could enhance its usability both as an educational resource and for day-to-day tasks.

Keywords: Data Modeling; Data Modeling Educational Tools; Data Modeling Education.

1. Introdução

No desenvolvimento de software, uma das etapas essenciais durante o estágio inicial de levantamento de requisitos e estruturação de soluções é a modelagem e criação de um banco de dados, capaz de suportar as informações necessárias para a validação e implementação das regras de negócio (AZEVEDO-JÚNIOR & CAMPOS, 2008).

Diversas técnicas e ferramentas são empregadas para simplificar o processo de modelagem de dados, sendo desenvolvidas com o propósito de definir estruturas padronizadas que possam ser revisadas em equipe e validadas com o usuário requisitante. Entre as ferramentas amplamente usadas destacam-se *MySql Workbench*[1], *Oracle SQL Developer Data Modeler*[2] e *DBSchema*[3], que oferecem um Sistema de Gerenciamento de Banco de Dados (SGBD) completo e uma base robusta para modelagem. São, geralmente, ferramentas preferidas por desenvolvedores experientes em fases de manutenção ou expansão de software (MEDEIROS; LIMA; RODRIGUES, 2018; FRESCURA, 2019).

No entanto, devido ao alto nível de personalização necessário para a manutenção abrangente de Bancos de Dados (BD), essas ferramentas costumam ser complexas, tornando-se desafiadoras para iniciantes. Para atender a essa demanda, surgem ferramentas com foco no ensino dessas tecnologias, que criam um ambiente menos intimidador e auxiliam na compreensão dos conceitos fundamentais de modelagem de dados. Um exemplo é o BRModelo[4], um projeto brasileiro desenvolvido em colaboração entre a Universidade Fe deral de Santa Catarina, Universidade Federal do Mato Grosso e o Instituto Mercado Livre (MELLO; CANDIDO; NETO, 2021).

O modelo de representação de dados JSON faz parte do ecossistema ECMAScript (ECMA, 2017) e é amplamente adotado por *Application Programming Interface* (API) para a representação de dados de BD para aplicações que tem como o objetivo o consumo desses dados para a uma interação com usuário através de componentes acessíveis como menus, campos,

[1] https://www.mysql.com/products/workbench/
[2] https://www.oracle.com/br/database/sqldeveloper/technologies/sql-data-modeler/
[3] https://dbschema.com/
[4] https://www.brmodeloweb.com/lang/pt-br/index.html

formulários e outros, denominadas *front-end*. Esse modelo já é explorado em mapeamentos e modelos de dados, conforme visto em trabalhos recentes (LEGUIZAMON, 2022; BOURHIS, REUTTER E VRGOC, 2020).

Neste trabalho, o objetivo foi o desenvolvimento do protótipo de uma ferramenta voltada à representação e modelagem de dados baseado em modelos de dados JSON, com a intenção de reaproveitar conhecimento pré-existente em ferramentas e técnicas de desenvolvimento *front-end*. Essa ferramenta foi apresentada na forma e uma CLI que recebe um arquivo JSON como entrada, e devolve instruções em DDL, através de um processo denominado transpilação, onde uma linguagem de programação é convertida para outra antes da sua execução, no caso presente, onde um modelo de dados final em SQL terá a mesma estrutura final do modelo de dados representado inicialmente em JSON. A ferramenta utiliza a sintaxe do motor de banco de dados PostgreSQL (POSTGRESQL, 2024) como base para transpilação dos comandos. O desenvolvimento oferece ao usuário controle sobre questões chave sobre o mapeamento, como tipos de *primary key* (PK), configuração de autoincremento e outros. Ela também oferece a opção de acompanhar o passo a passo da transpilação e solicitar um resumo do resultado da análise da estrutura JSON.

Este artigo, está dividido em seções, sendo a próxima Seção, Referencial Teórioco, essa seção busca explorar as bases teóricas dos fundamentos e ferramentas utilizados durante o desenvolvimento dessa pesquisa. Na Seção 3, Estado da Arte, serão expostas as pesquisas já existentes na área em busca de encontrar lacunas ao qual esse trabalho pode cobrir. A Seção 4 Metodologia, são enumerados os métodos os quais foram seguidos para implementação da ferramenta proposta. A Seção de resultado tem como objetivo, expor os resultados obtidos através da execução dessa pesquisa, e de seus artefatos. A Seção 6, Considerações Finais, avalia os resultados de acordo com os objetivos levantados, e elenca pontos de melhoria observados através dessa análise comparativa.

2. Referencial Teórico

Durante o desenvolvimento deste estudo, foram aplicados conhecimentos específicos da área de Engenharia de Software, os quais serão descritos nas subseções a seguir.

2.1. O modelo de representação de dados JSON

A *Ecma International*, define JSON como um modelo de representação de estado de dados e foi padronizado pela mesma em 2013, com posterior atualização em 2017 (ECMA, 2013; ECMA, 2017).

A *Ecma* é uma associação que tem como o objetivo a padronização de sistemas de informação e comunicação, neste sentido, a associação realiza reuniões periódicas com a presença de representantes de diferentes empresas de tecnologia do mundo, como *Microsoft®, Apple®, IBM®, Google®, Meta®* e outras. (ECMA, 2024).

Nessas reuniões são discutidas e definidos padrões que serão aplicados nas empresas participantes da associação, com o objetivo de diminuir o retrabalho, e aumentar a interoperabilidade de soluções tecnológicas em diferentes produtos independente de sua empresa.

O modelo JSON (Figura 1) é um texto composto de uma sequência de caracteres que podem representar estruturas complexas de dados utilizando chaves, colchetes, pontos-duplos, vírgulas e aspas, além de seu conteúdo variado que pode ser qualquer caractere Unicode (ECMA, 2017).

Figura 1 - Exemplo de estrutura JSON

```json
{
  "cargo": "programador",
  "data_inicio": "2020-01-01",
  "tempo_de_casa": 25,
  "salario": 1000.2,
  "pessoa": {
    "nome": "João",
    "sobrenome": "Silva",
    "idade": 30
  },
  "filho": [
    {
      "nome": "Maria",
      "sobrenome": "Silva",
      "idade": 25
    },
    {
      "nome": "José",
      "sobrenome": "Silva",
      "idade": 30
    }
  ]
}
```

Fonte: Autor, 2024

JSON é modelo padrão de representação de estado em ambientes de desenvolvimento e consumo de APIs que seguem o design *Representional State Transfer* (MARRS, 2017), porém esse não é o único, podendo também ser representados através de outros modelos como XML, YAML e outros.

2.2. Modelagem de dados

A fase de modelagem de dados é uma das mais importantes do desenvolvimento inicial de uma aplicação, pois é o mapeamento efetivo de conceitos do mundo real em representações no mundo virtual, e garante a contextualização das informações salvas pela aplicação no banco de dados. A falta dessa contextualização torna toda informação salva nele sem valor (BRANDT, 2024).

Dentre as estratégias utilizadas para modelar esses dados, podemos destacar o desenvolvimento do modelo Entidade-Relacionamento (ER), proposto por Cheng (1976) que define conceitualmente cada parte do mundo real como uma entidade, e a forma como essas entidades se relacionam, chamados relacionamentos.

Os relacionamentos são representados por suas entidades envolvidas, e sua semântica, isto é, sua cardinalidade, que pode ser um para um (1:1), um para muitos (1:n) e muitos para muitos (m:n), que representam a quantidade em que essas entidades se relacionam, como por exemplo a representação de um imóvel e seu proprietário, é representada pela relação 1:n em que um imóvel pode ter apenas um proprietário, mas um proprietário pode ter vários imóveis. Dessa forma é possível entender as dependências dos relacionamentos entre as entidades, e compreender quais entidades dependem da existência das suas contrapartes.

Dessa forma, o modelo proposto por Cheng (2017) representa de maneira conceitual o modelo de dados de forma que pessoas técnicas e não técnicas conseguem interpretá-las e discuti-las para esclarecimentos e levantamento de requisitos.

2.3. Command Line Interfaces

As CLI são ferramentas populares entre desenvolvedores e usuários avançados de computadores, devido ao seu alto nível de controle de parâmetros e detalhamento de progresso, com um menor tempo de configuração, com exemplares

bastante difundidos na comunidade de desenvolvimento de software, como por exemplo Git[5], Docker[6], Node[7]. (SAMPATH, MERRICK, MACVEAN, 2021).

Devido ao seu maior nível de personalização, e refinamento dos parâmetros passados para a CLI, muitos usuários tendem a evitar sua utilização pela necessidade de ter um alto conhecimento de sintaxe para maior controle sobre os parâmetros e consequentemente, maior aproveitamento da ferramenta (MURILLO, SÁNCHEZ, 2014).

Grande parte dos maiores provedores de computação em nuvem como *Azure, Amazon Web Services (AWS), Google Cloud Platform (GCP)*, e outras, disponibilizam CLIs para interação com seus ambientes em nuvem, com opções de criação de instâncias de serviços, controle de acessos e monitoramento (MURILLO, SÁNCHEZ, 2014). Também é comum aos motores de bancos de dados disponibilizarem versões de controle de dados em linha de comando, como por exemplo *psql* do PostgreSQL*, oci* da Oracle*, mysql* do MySQL.

2.4. Visual Studio 2022

Visual Studio é uma *Integrated Development Environment* (IDE) da Microsoft com foco no desenvolvimento de aplicações no ecossistema .NET. É uma ferramenta poderosa de codificação, estruturação, e depuração de projetos que fazem parte desse ecossistema (MICROSOFT, 2024c)

A ferramenta oferece ao desenvolvedor controle de todo o projeto, como criação de soluções (conjuntos de projetos), projetos, controle de bibliotecas e de ferramentas e linguagens instaladas. Além disso, a IDE conta com *Intellisense* para todas suas linguagens suportadas, templates de arquivos como classes, enumeradores, e integração com o banco de dados SQL Server.

Essa ferramenta foi selecionada para acelerar o processo de desenvolvimento da aplicação, diminuir o erro humano, e facilitar o processo de criação de casos teste e depuração de fluxo de informação na aplicação.

---

[5] https://git-scm.com/
[6] https://www.docker.com/
[7] https://nodejs.org/pt

2.5. DotNet Core 8.0

É o ecossistema de desenvolvimento da Microsoft, que proporciona fácil acesso às bibliotecas padrão, compilação multiplataforma, bibliotecas de consultas de dados, suporte a injeção de dependência nativa, fácil publicação da aplicação e disponibilização através das lojas da Microsoft (MICROSOFT, 2024a).

2.6. Csharp (C#)

Linguagem de programação da Microsoft utilizada no ecossistema .NET e no Mono, utilizado pela *game engine* Unity. É uma linguagem baseada no paradigma de orientação a objetos, portanto, conta com alto nível de detalhamento de tipos, hierarquias e implementações (MICROSOFT, 2024b).

2.7. Spectre Console Cli

Biblioteca para desenvolvimento de aplicações do tipo CLI com suporte para controle de cores de saídas, criação de múltiplos comandos, definição de argumentos obrigatórios e opcionais, definição de parâmetros extras e injeção de dependência própria (SVENSSON, SCOTT, ANDRESSEN, 2024).

2.8. PostgreSQL

Banco de dados *open source*, gratuito, com suporte para tipos de dados existentes em outros motores de banco de dados, com sintaxe não muito diferente da sintaxe “crua” do SQL (POSTGRESQL, 2024).

2.9. DBeaver

Ferramenta *open source* de visualização de dados e diagramas de relacionamento entre tabelas de bancos de dados. Disponível gratuitamente, ela tem suporte para a maioria dos motores de bancos de dados disponíveis no mercado (DBEAVER, 2024).

3. Estado da Arte

Essa seção apresentará estudos realizados na englobando os temas que esse estudo se propõe a tratar, de forma a encontrar lacunas que possam ser preenchidas pelo desenvolvimento do mesmo. Essa seção foi desenvolvida na forma de uma revisão de literatura exploratória desenvolvida a partir dos bancos de dados Google

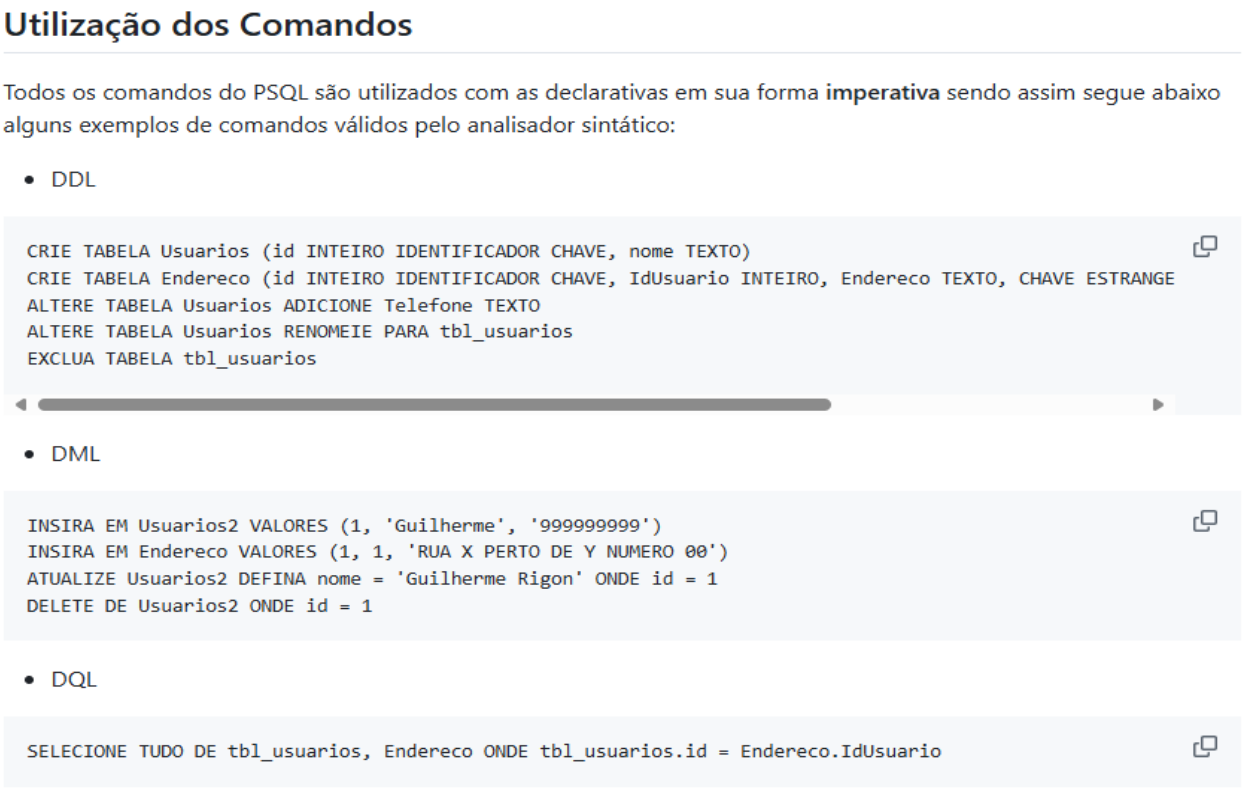
Acadêmico e *ResearchGate* no período de 2020 a 2024, as palavras-chaves foram: Ferramentas de Modelagem de dados, JSON, JSON Schema, JSON SQL

3.1. Ferramentas de ensino de modelagem de dados baseadas em bancos de dados relacionais

O trabalho de Xing, Salem e Ling (2024) apresenta o *Learning Environment and Resource Network for Databases* (LEARNDB) uma plataforma educacional para o ensino de tecnologias de banco de dados. A plataforma busca oferecer conteúdo sobre todo o ecossistema de banco de dados, desde sua modelagem até sua manipulação através de SQL, a plataforma tem espaço dedicado a professores para criação e controle de aulas e exercícios.

No trabalho Rigon, Oliveira e Vidal (2022) é apresentada uma ferramenta de ensino que permite o aluno interagir com conceitos de *Data Manipulation Language* (DML) e *Data Querying Language* (DQL) em português chamada Valkyrie. Inspirado em Portugol, uma linguagem de programação que permite que o aluno se familiarize primeiro com os conceitos de programação sem se preocupar com os comandos, a ferramenta oferece ao aluno a oportunidade de se familiarizar primeiramente com os conceitos de manipulação e seleção para depois se aprofundar em seu motor de banco de dados de escolha. A ferramenta é disponibilizada na forma de CLI e exemplos fornecidos pela sua documentação do *github* são exibidos na Figura 2.

Figura 2 - ValkyrieDb Documentação no Github[8]



Utilização dos Comandos

Todos os comandos do PSQL são utilizados com as declarativas em sua forma **imperativa** sendo assim segue abaixo alguns exemplos de comandos válidos pelo analisador sintático:

- DDL

```
CRIE TABELA Usuarios (id INTEIRO IDENTIFICADOR CHAVE, nome TEXTO)
CRIE TABELA Endereco (id INTEIRO IDENTIFICADOR CHAVE, IdUsuario INTEIRO, Endereco TEXTO, CHAVE ESTRANGE
ALTERE TABELA Usuarios ADICIONE Telefone TEXTO
ALTERE TABELA Usuarios RENOMEIE PARA tbl_usuarios
EXCLUA TABELA tbl_usuarios
```

- DML

```
INSIRA EM Usuarios2 VALORES (1, 'Guilherme', '999999999')
INSIRA EM Endereco VALORES (1, 1, 'RUA X PERTO DE Y NUMERO 00')
ATUALIZE Usuarios2 DEFINA nome = 'Guilherme Rigon' ONDE id = 1
DELETE DE Usuarios2 ONDE id = 1
```

- DQL

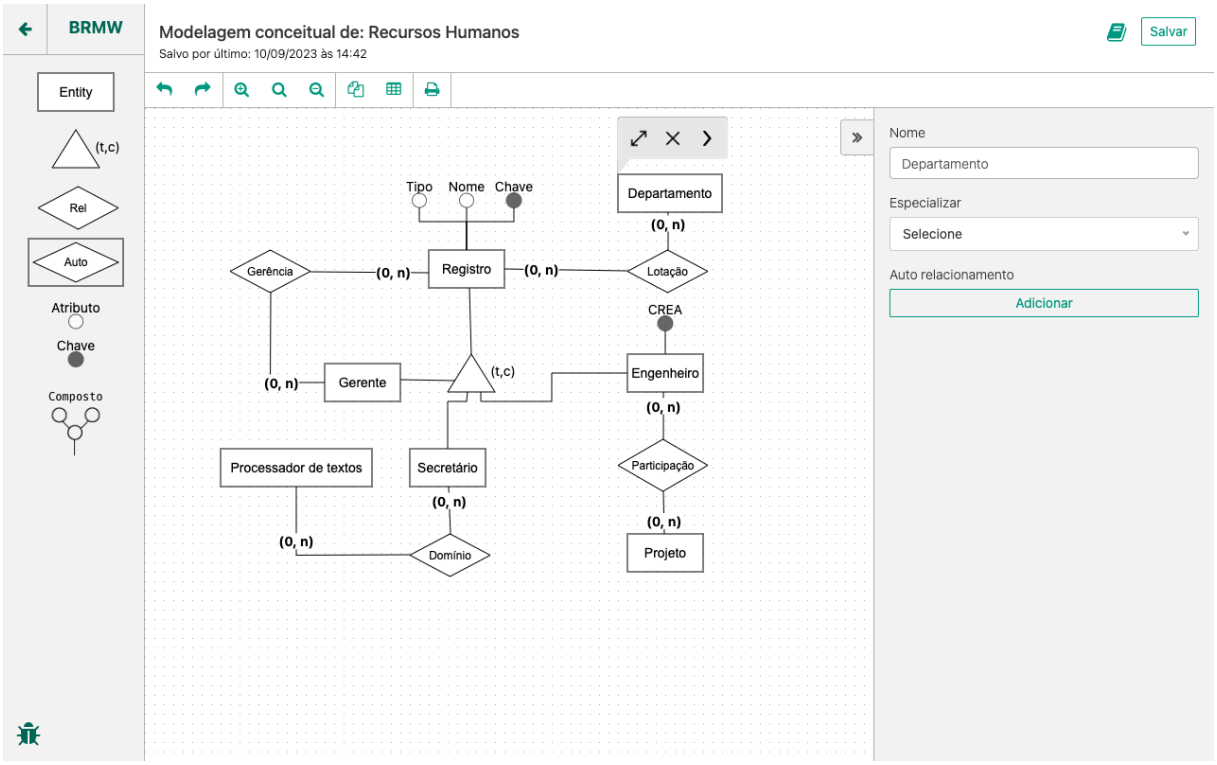
```
SELECIONE TUDO DE tbl_usuarios, Endereco ONDE tbl_usuarios.id = Endereco.IdUsuario
```

Fonte: GitHub - Guilherme Rigon, 2022

A ferramenta BRModelo é apresentada no trabalho de Mello, Cândido e Neto (2021) existente desde 2005, a ferramenta vem recebendo atualizações ao longo dos anos e é comumente utilizada em ambiente de ensino universitário. Com o objetivo de ser uma ferramenta intuitiva apresentada na Figura 3 e com capacidade de gerar modelos conceituais e lógicos e apresenta versão *web*.

[8] https://github.com/Guilherme-Rigon/ValkyrieDB

Figura 3 - Interface BRModelo Web



Fonte: BRModelo, 2024

3.2. Ferramentas que utilizam JSON e estruturas de dados

No trabalho de Leguizamon (2022), foi desenvolvida uma ferramenta com o objetivo de fazer a migração da estrutura de bancos de dados *Not Only SQL* (NOSQL) baseados em documentos, para bancos de dados NOSQL baseados em grafos. A solução foi criada utilizando uma arquitetura baseada em eventos, e teve desempenho superior ao plugin do provedor que banco de dados de grafos disponibiliza para realizar a mesma operação.

A pesquisa realizada por Padilha (2020) apresenta um método para casamento de documentos *JSON* baseados com o objetivo de identificar estruturas similares em diferentes níveis da árvore e juntá-las como similares. O método apresentado contém passos automatizados por algoritmos e passos manuais. As fases automatizadas foram desenvolvidas em Java e a ferramenta se demonstrou eficiente segundo a autora.

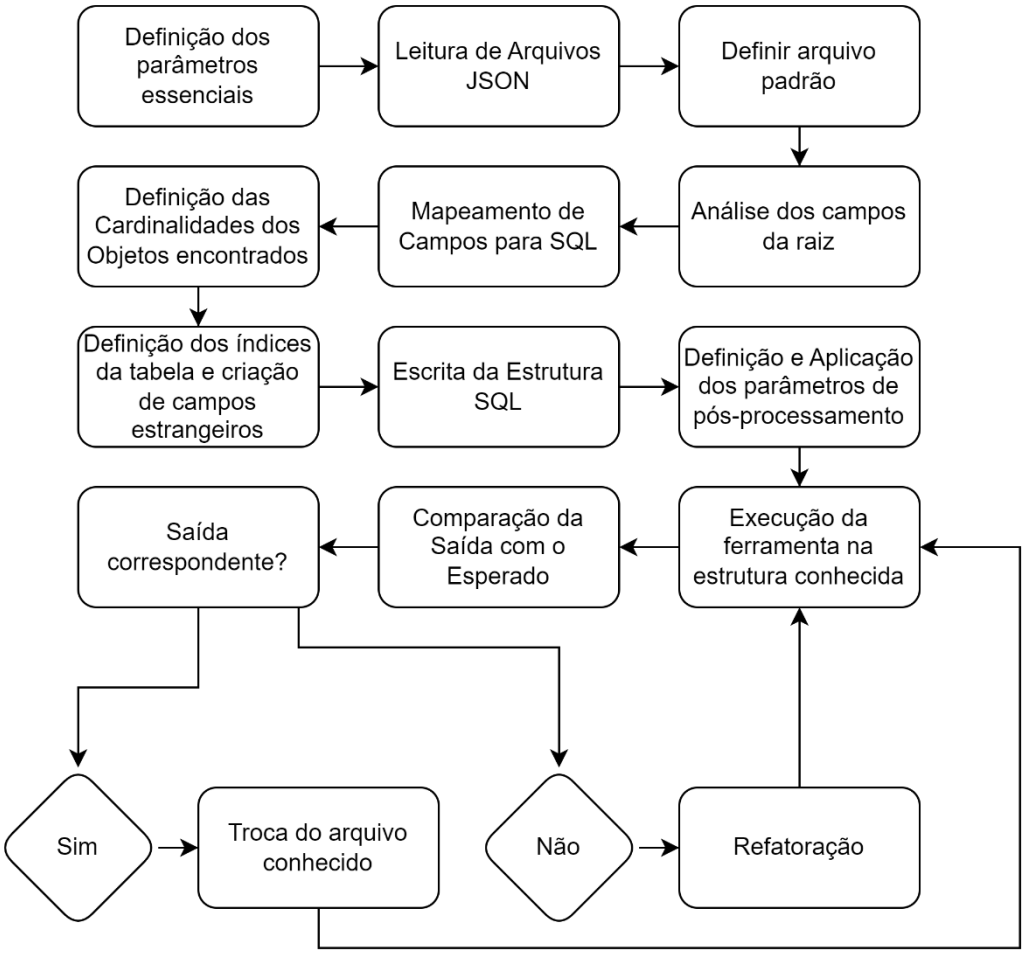
A análise das ferramentas disponíveis demonstra que ferramentas de ensino de modelagem de dados vêm sendo estudadas e desenvolvidas. Portanto, é possível

perceber espaço para uma ferramenta utilizando o modelo de representação de dados JSON, que já conta com estudos na área de modelagem de dados.

4. Metodologia

A aplicação foi desenvolvida sequencialmente num fluxo de desenvolvimento que foi dividida nas etapas apresentadas na Figura 4, e detalhadas em seguida:

Figura 4 - Fluxograma de Desenvolvimento



Fonte: Autor, 2024

4.1. Definição dos parâmetros da CLI

Nessa fase, foram definidos os principais parâmetros presentes na CLI, essenciais para o cumprimento da atividade proposta.

4.2. Leitura dos arquivos JSON

Nessa etapa, foi implementada a lógica da leitura dos arquivos JSON fornecidos pelo usuário, por meio de um *path* para o arquivo de interesse.

4.3. Carregamento de arquivo padrão

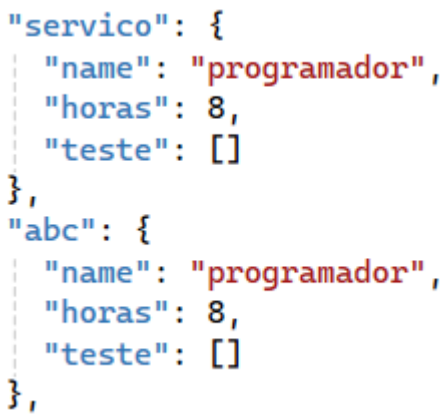
Após os passos anteriores, foi definido um arquivo padrão para ser usado como leitura em que seu conteúdo é conhecido, e sua saída esperada também é conhecida.

4.4. Análise dos campos iniciais do JSON

Nessa fase, os campos encontrados na raiz do arquivo JSON são avaliados, sendo separados em campos simples, que são aqueles que já têm seu tipo de dado definido nesse nível, e campos complexos, aqueles que descem um nível na árvore.

Ainda nessa fase, os campos complexos são comparados entre si, e se encontrada alguma similaridade entre eles, eles serão agrupados para que sejam representados em uma única tabela, exemplificado através da Figura 4.

Figura 5 - Exemplo campos similares a serem agrupados



Fonte: Autor, 2024

4.5. Mapeamento dos Campos JSON para SQL

Nesse momento todas as possíveis tabelas são verificadas internamente quanto aos seus nomes e tipos de campos, todas as conversões são feitas levando em consideração apenas uma sintaxe de motor de banco de dados: PostgreSQL.

4.6. Definição de Cardinalidades dos Objetos

As cardinalidades das tabelas envolvidas nas operações são todas relacionadas a tabela que representará a estrutura raiz, dessa forma, objetos ou *arrays* em um nível de árvore menor que o do primeiro imediatamente abaixo da raiz não serão computados.

4.7. Definição e criação dos índices das tabelas

Após as cardinalidades serem definidas, esses dados serão utilizados para popular informações sobre os índices que serão criados para cada uma das tabelas, criando os campos necessários nas tabelas corretas e adicionando as *constraints* necessárias.

4.8. Escrita de Estrutura em SQL

Após a finalização do mapeamento dos campos e dos índices, a aplicação transcreve essa estrutura para SQL. Primeiramente são criadas todas as tabelas com todos os campos e todas as PKs e depois as tabelas são alteradas para a criação das constraints de *FOREIGN KEY* (FK). Dessa forma, garantindo que todas as PKs estivessem presentes durante a criação das FKs relacionadas.

4.9. Aplicação dos parâmetros definidos

Nesse momento do desenvolvimento foram adicionados mais parâmetros para a execução e o controle dos fluxos da aplicação, e oferecer ao usuário mais opções de personalização da ferramenta e do nível de detalhe das saídas desejadas.

4.10. Testes

Os testes foram baseados em uma série de arquivos JSON, que tinham conteúdo, e estrutura de banco conhecidas, estes arquivos foram utilizados como input para a aplicação, e o resultado encontrado, comparado com a estrutura de banco de dados já conhecida para avaliar a similaridade entre as estruturas. Estes testes foram realizados de maneira cíclica durante o desenvolvimento a fim de abranger todos os cenários propostos pela aplicação. Os arquivos utilizados durante o processo de testes de mapeamento estão disponíveis no GitHub do projeto[9].

[9] https://github.com/1baldasso/jtsql

4.11. Refatorações

Ao longo do ciclo de desenvolvimento, ao executar algum dos testes descritos na subseção anterior e alguma inconformidade ser encontrada, a área responsável por gerar a inconformidade foi avaliada, corrigida, e reiniciado o ciclo de testes.

5. Resultados

Os parâmetros definidos como essenciais para a fase de definição de parâmetros foram: *path, output-path, root-name* e *key.*

A leitura do arquivo JSON foi feita através do parâmetro path recebido como argumento pela CLI. Quando não é informado argumento, um arquivo padrão é selecionado. Mostrado pela Figura 6.

Figura 6 - Utilização de arquivo padrão sem informar um arquivo inicial

```
PS D:\TCC\JsonToSql\JsonToSql.CLI\bin\Release\net8.0> jtsql -s
CREATE TABLE IF NOT EXISTS root (
    root_id uuid PRIMARY KEY,
    cargo varchar,
    data_inicio date,
    tempo_de_casa integer,
    salario decimal,
    pessoa_root_id uuid
);

CREATE TABLE IF NOT EXISTS filho (
    filho_id uuid PRIMARY KEY,
    nome varchar,
    sobrenome varchar,
    idade integer,
    filho_root_id uuid,
    root_pessoa_id uuid
);

ALTER TABLE root
ADD CONSTRAINT FK_filho_root_filhofilho_id_pessoa_id FOREIGN KEY (pessoa_root_id) REFERENCES filho (filho_id);

ALTER TABLE filho
ADD CONSTRAINT FK_root_filho_rootroot_id_filho_id FOREIGN KEY (filho_root_id) REFERENCES root (root_id),
ADD CONSTRAINT FK_root_filho_rootroot_id_pessoa_id FOREIGN KEY (root_pessoa_id) REFERENCES root (root_id);
```

Fonte: Autor, 2024

A estrutura do arquivo JSON padrão é conhecida, de foram que uma saída válida já é conhecida antes do desenvolvimento da aplicação (Figura 7).

Figura 7 - Estrutura do arquivo JSON padrão

```json
{
  "cargo": "programador",
  "data_inicio": "2020-01-01",
  "tempo_de_casa": 25,
  "salario": 1000.2,
  "pessoa": {
    "nome": "João",
    "sobrenome": "Silva",
    "idade": 30
  },
  "filho": [
    {
      "nome": "Maria",
      "sobrenome": "Silva",
      "idade": 25
    },
    {
      "nome": "José",
      "sobrenome": "Silva",
      "idade": 30
    }
  ]
}
```

Fonte: Autor, 2024

Essa estrutura considera uma saída válida conhecida a representada na tabela a seguir onde “agrupamento” se refere aos campos complexos em que as estruturas que podem ser representadas por uma única tabela, que pode ser nomeada com um dos nomes agrupados:

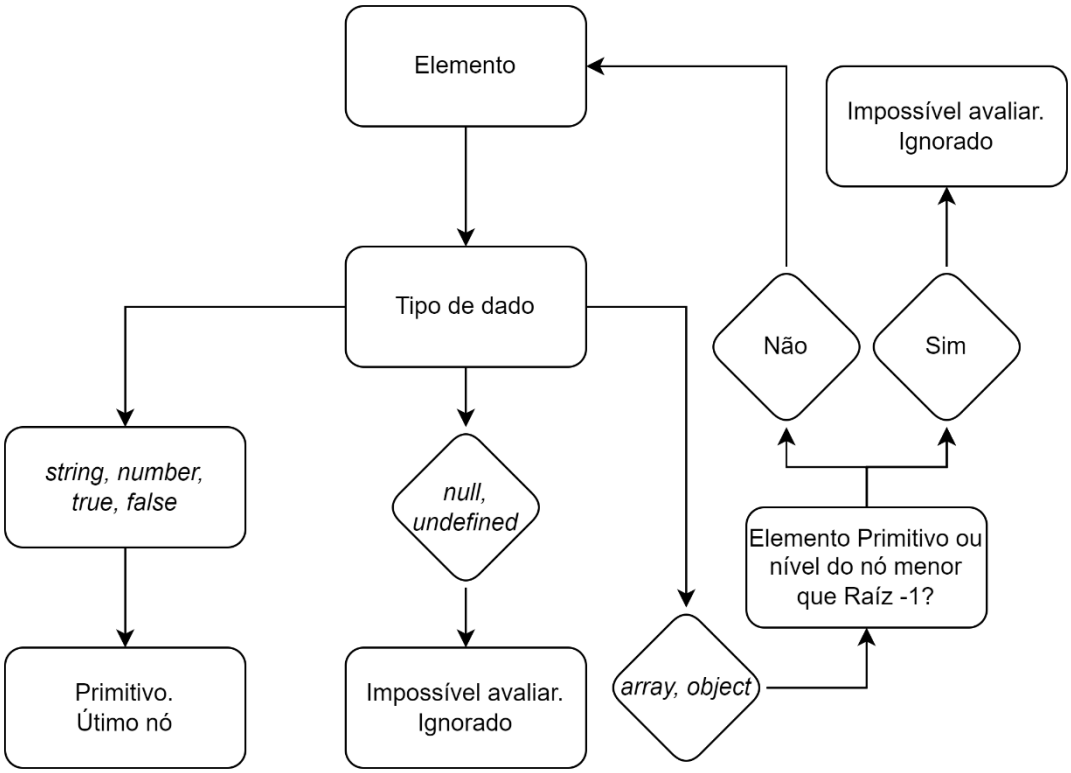
Tabela 1 - Mapeamento esperado do arquivo conhecido

Campos simples		**Campos Complexos**		
Nome	**Tipo**	**Nome**	**Tipo**	**Agrupamento**
cargo	Texto	pessoa	Tabela	pessoa; filho;
data_inicio	Data	filho	Tabela	pessoa; filho;
tempo_de_casa	Inteiro			
salario	Decimal			

O método de extração de tipo da estrutura JSON fornecida foi através da propriedade *ValueKind* da estrutura *JsonElement*, padrão de representação de campos JSON em .NET. Essa estrutura foi avaliada quanto aos seus tipos primitivos

e separada seguindo o seguinte padrão, campos simples com tipo *string, number, true* e *false* foram considerados campos simples, seus nós finais. Os campos do tipo *null* e *undefined* não são mapeados pois não há conversão direta, os campos do tipo *object* e *array* descem um nível na árvore, sendo consideradas estruturas complexas, e, portanto, passando por uma segunda fase de validação um esquema da primeira fase é apresentado na Figura 8.

Figura 8 – Esquema de análise de elementos JSON



Fonte: Autor, 2024

Na segunda fase do mapeamento, as estruturas complexas são avaliadas, sendo que as *arrays* de campos primitivo não são mapeadas, devendo ser informados objetos caso a intenção seja a criação uma relação 1:n.

Nessa etapa também são definidas as cardinalidades dos campos complexos encontrados, as cardinalidades são definidas da seguinte forma:

- Caso o valor avaliado seja uma *array*, é verificado se na definição dos objetos internos desse vetor, algum dos campos tem o nome do campo raiz e é um

vetor, se sim, a estrutura será considerada m:n, caso contrário 1:n com o lado forte sendo a raiz.

- Caso o valor avaliado seja um *object*, é verificado se esse objeto contém algum valor que seja um vetor com um nome similar ao da tabela raiz, ele é considerado 1:n com o lado forte sendo o objeto avaliado, caso contrário a relação é considerada sendo 1:1.

Após a definição dos candidatos a tabelas, e a definição das cardinalidades, as tabelas são mapeadas quanto aos seus campos internos seguindo o método já definido anteriormente para mapeamento de campos simples. Todas as tabelas são então enviadas para fase de análise de campos e criação de índices, nessa fase os campos são avaliados e com base nas cardinalidades definidas, novos campos são criados para representar as devidas *PKs* e *FKs* o tipo desses campos é controlado pelo parâmetro *–key* que define uma chave como *varchar, integer* ou *uuid*. O Código responsável pela geração de um índice 1:n é representado pela Figura 9.

Figura 9 - Algoritmo responsável pela Criação de Índices 1:n

```
var fieldId = new DatabaseFieldInfo
{
    Name = $"{field.Name}_{root.Table.Name}_id",
    ClrType = rootPk.ClrType,
    DatabaseType = rootPk.DatabaseType == "serial" ? "integer" : rootPk.DatabaseType,
    IsAutoIncrement = false,
    IsNullable = true,
};
c.Table.Fields.Add(fieldId);
c.Table.Indexes.Add(new DatabaseIndex
{
    IsForeignKey = true,
    ReferencesField = root.Table.Indexes.FirstOrDefault(x => x.IsPrimaryKey).Field.Name,
    ReferencesTable = root.Table.Name,
    Field = fieldId,
    Name = $"FK_{root.Table.Name}_{c.Table.Name}_{root.Table.Name}{rootPk.Name}_{field.Name}_id",
});
```

Fonte: Autor, 2024

A partir do mapeamento e nomeação de índices, a aplicação segue para a fase de escrita de SQL, onde primeiro são definidas todas as tabelas, campos e *PKs* e FKs envolvidas na instrução *CREATE TABLE,* sem adicionar as *CONSTRAINTS* necessárias. Em seguida são preparadas instruções *ALTER TABLE* que alteram todas as tabelas que contém relacionamentos adicionando suas respectivas CONSTRAINTS.

Após a transcrição do SQL, ele é guardado em memória para que os parâmetros de pós processamento sejam aplicados. Caso o parâmetro *–describe* seja passado, ele insere uma descrição das cardinalidades encontradas antes da transcrição do SQL, em forma de comentário. Caso o parâmetro *–output-path* seja enviado, um arquivo é criado no destino com o resultado da transpilação. O parâmetro *–output-console* define se a resposta será exibida também na saída do console utilizado. O parâmetro *–copy-output* copia o resultado da transpilação para a área de transferência. O parâmetro *–verbose* é utilizado durante todo o fluxo da aplicação e cria um log detalhado de cada passo tomado pela transpilação em ordem.

A interface da ferramenta desenvolvida conta com lista de comandos disponíveis para que usuário saiba os parâmetros que podem ser controlados. Essa lista está descrita em inglês por se tratar do idioma padrão de muitas ferramentas disponíveis atualmente, a Figura 10 exibe a forma como essa lista é apresentada ao usuário.

Figura 10 - Lista de comandos. Acessível através do parâmetro *-h*

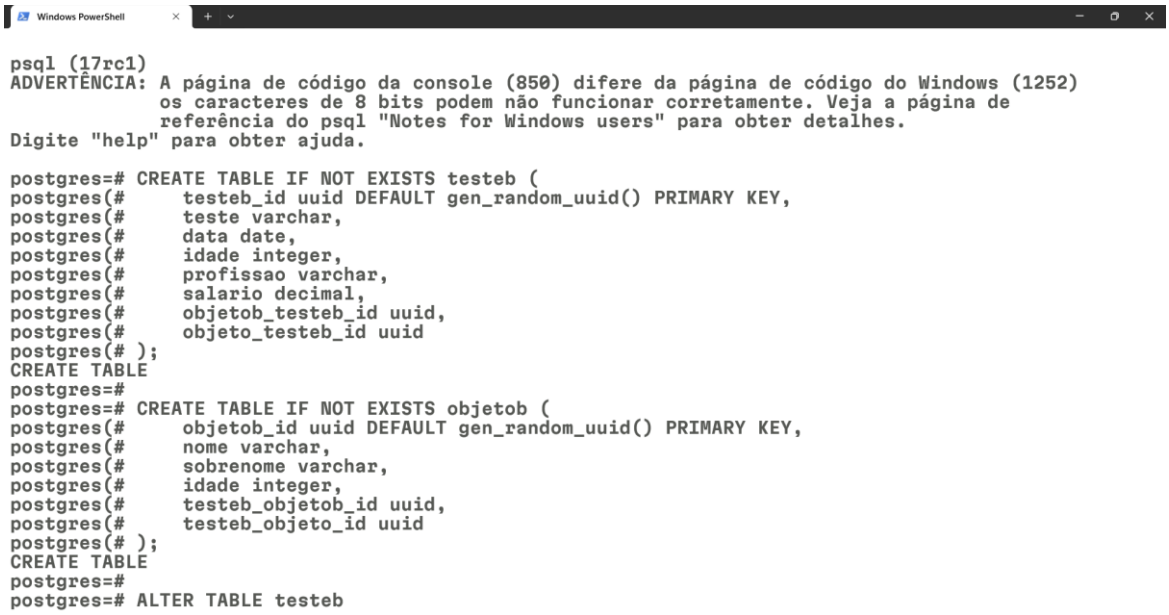
```
PS C:\Users\lucas> jtsql -h
USAGE:
    jtsql.dll        [OPTIONS]

ARGUMENTS:
              The path of the JSON file containing the structure to translate

OPTIONS:
                            DEFAULT
    -h, --help                          Prints help information
    -n, --root-name                     Defines the table name of the transpiled data found in the root JSON object.
                        [root-name]     The default value is the json FileName or 'root' when no file is
                                        provided
    -o, --output                        The path in which the translated SQL DDL query will be written to
    -k, --key                 uuid      Defines the identifier type. Valid values: uuid | integer | int | varchar
    -s, --output-console                Outputs the translated content into the console
    -i, --interactive                   Initialize an interactive version of the program where you can write your JSON
                                        manually (beta)
    -d, --describe                      Describe what has been done to get to this result commented on the sql file and
                                        on screen if console-output selected
    -v, --verbose                       Logs every operation status to the console
    -c, --copy-output                   Copies the output into the clipboard
    -a. --auto-increment                Defines if the key auto-increments
```

Fonte: Autor, 2024

A leitura e interpretação dos parâmetros passados para o programa através da CLI foram consistentes ao longo do desenvolvimento e dos testes. Os diagramas avaliados correspondem a uma estrutura que comportaria todos os dados recebidos através do JSON, as explicações fornecidas através do comando *--describe* são úteis para o entendimento da estrutura do banco de dados resultante. As funcionalidades que interagem diretamente com a experiência do usuário com a saída do comando como *--copy-output*, *--output-console* e *--output* melhoraram a experiência geral do

usuário e abrem um leque de possibilidades utilizando operadores de transferência de saída de comando para entrada do Linux, podendo a saída do comando da CLI ser utilizada diretamente como argumento da entrada de um outro CLI como por exemplo *mysql -u user -p database_name $(jtsql ./teste.json -s).* A Figura 11 mostra a execução das instruções geradas através da CLI interativa *psql* para a criação das tabelas do banco de dados.

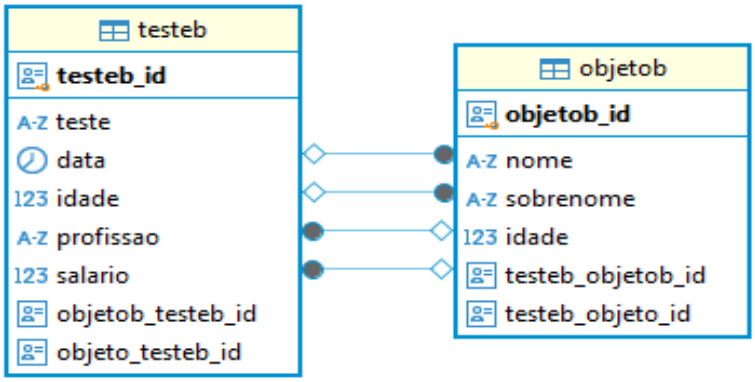
Figura 11 - Executando Código de saída no psql



Fonte: Autor, 2024

Após a execução das instruções através do psql, é possível verificar o diagrama gerado pelo DBeaver (Figura 12) através da sua interface gráfica. Esse diagrama nos mostra que as relações esperadas foram geradas corretamente.

Figura 12 - Diagrama do banco de dados gerado pelas instruções geradas pela ferramenta



Fonte: Autor, 2024

6. Considerações Finais

O protótipo foi desenvolvido com sucesso, sendo que a leitura de diversos arquivos JSON foi realizada com sucesso, e a resposta da aplicação foram instruções DDL válidas que geram tabelas que comportam os dados contidos no JSON.

Os comandos definidos alteraram o resultado de forma satisfatória, se mantendo ainda no esperado previamente, dando ao usuário controle sobre questões como nome da tabela da raiz, tipo de dado dos ids, autoincremento e outros.

Não foi possível avaliar o impacto da aplicação como ferramenta de ensino, porém, através desse trabalho, foi possível concluir que existem lacunas que podem ser preenchidas por uma ferramenta para ensino e modelagem de dados com base em modelos de dados JSON por se tratar de um possível conhecimento prévio já presente em alunos que já tem algum conhecimento em tecnologias *front-end* e em JSON.

Ao desenvolver a ferramenta foi possível concluir que ela pode ser eficiente para casos de demonstração de como os dados presentes em um JSON podem estar sendo representados em um banco de dados relacional. Mas a ferramenta apresenta limitações quanto a personalização individual dos campos como precisão, tamanho máximo das *strings*, perda de dados escondidos ou anulados, portanto, ainda não é uma ferramenta suficiente para ser utilizada em ambientes de fidelidade máxima dos bancos de dados.

Ao reconhecer as deficiências da ferramenta podemos concluir que sua aplicação se estende apenas ao campo das demonstrações e conceituações. Os próximos passos desta pesquisa são:

- Melhoria de versão interativa;
- Maior personalização dos campos individuais;
- Habilitar mapeamento de níveis mais inferiores;
- Preparação para uma fase beta;
- *Wizard* de instalação da ferramenta;
- Feedback de usuários;

A execução desses passos continuará através do repositório no GitHub[10].

[10] https://github.com/1baldasso/jtsql

Referências

AZEVEDO-JÚNIOR, DP.; CAMPOS, R. Definição de requisitos de software baseada numa arquitetura de modelagem de negócios. **Revista Brasileira de Engenharia de Software,** v. 18, n. 1, p. 26-46, 2008.

BRANDT, M.P.; VIDOTTI S.A.B.G. Arquitetura da informação para processos de negócio e modelagem de banco de dados: aproximações possívei**s**. v. 30, e-131304. **Em Questão**, 2024.

BOURHIS, PIERRE; *et al*. JSON: Data model and query languages. **Information Systems**, v. 89, 2019.

BRMODELO. **BR Modelo Web - Ferramenta online para modelagem de banco de dados**, [s.d.]. Disponível em: https://www.brmodeloweb.com/lang/pt-br/index.html/. Acesso em 31 out. 2024.

CHENG P.P.S. The entity-relationship model—toward a unified view of data. **ACM Transactions on Database Systems**, v. 1, p 9-36. 1976.

DBEAVER **DBeaver User Guide**, [s.d.]. v 24.3.ea. Disponível em: https://dbeaver.com/docs/dbeaver/Getting-started/. Acesso em: 31 out. 2024.
Acesso em: 31 out. 2024

ECMA, INTERNATIONAL. **ECMA-404: The JSON Data Interchange Format**. Documento Eletrônico. Geneva: ECMA, 2013. Disponível em https://www.ecma-international.org/wp-content/uploads/ECMA-404_1st_edition_october_2013.pdf/.
Acesso em: 31 out. 2024.

ECMA, INTERNATIONAL. **ECMA-404: The JSON Data Interchange Syntax**. Documento Eletrônico. Geneva: ECMA, 2017. Disponível em https://www.ecma-international.org/wp-content/uploads/ECMA-404_2nd_edition_december_2017.pdf/.

LEGUIZAMON, P. V. **Uma Proposta para Mapeamento de Documentos JSON para Bancos de Dados de Grafos de Propriedades.** Trabalho de Conclusão de Curso do

Curso de Sistemas de Informação do Departamento de Informática e Estatística da Universidade Federal de Santa Catarina, Florianópolis, 2022.

MARRS, T. **JSON at Work: Practical Data Integration for the Web**. 1. ed. Sebastopol: O'Reilly, 2017.

MELLO, R.S.; CÂNDIDO, C. H.; BITTENCOURT S. NETO, M. brModelo: An Initiative for Aiding Database Design. **Journal of Information and Data Management**, [S. l.], v. 12, n. 2, 2021. DOI: 10.5753/jidm.2021.1983. Disponível em: https://sol.sbc.org.br/journals/index.php/jidm/article/view/1983/. Acesso em: 15 abr. 2024.

MICROSOFT **Baixar .NET**, © 2024a. Disponível em: https://dotnet.microsoft.com/pt-br/download/. Acesso em 31 out. 2024.

MICROSOFT **Novidades no C# 12**, © 2024b. Disponível em: https://learn.microsoft.com/pt-br/dotnet/csharp/whats-new/csharp-12/. Acesso em 31 out. 2024.

MICROSOFT **Visual Studio IDE 2022 – Ferramenta de Programação para Desenvolvedores de Software**, © 2024c. Disponível em https://visualstudio.microsoft.com/pt-br/vs/. Acesso em 31 out. 2024.

PADILHA, RJ. **Um processo para casamento de esquemas de documentos JSON baseado na estrutura e nas instâncias**. Dissertação apresentada ao Programa de Pós-graduação em Ciência da Computação da Universidade Federal de Santa Maria, Santa Maria – RS, 2020.

SIMÕES RIGON, GL; DE OLIVEIRA, M A; VIDAL, LF. **Aprendizado e funcionamento de banco de dados: Valkyriedb**. Episteme Transversalis, [S.l.], v. 13, n. 2, 2022. ISSN 2236-2649. Disponível em: https://revista.ugb.edu.br/index.php/episteme/article/view/2628/. Acesso em: 02 nov. 2024.

STACK OVERFLOW. **2023 Developer Survey**. 2023. Disponível em: https://survey.stackoverflow.co/2023/. Acesso em: 15 abr. 2024.

SVENSSON P.; SCOTT P.; ANDRESSEN N. **Spectre.Console – Welcome!**, © 2024. Disponível em: https://spectreconsole.net/. Acesso em: 31 out. 2024.

THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. **PostgreSQL: About**, © 1996-2024. Disponível em: https://www.postgresql.org/about/. Acesso em 31 out. 2024.

XING, G, et al. LEARNDB: A Comprehensive Toolkit for Database Education. In **Proceedings of the 55th ACM Technical Symposium on Computer Science Education** V. 2 (SIGCSE 2024), 1861–1862.