Evolução Incremental de Esquemas de Banco de Dados Orientado a Documentos

Eleonilia Monteiro Rodrigues[1], **Carlos Eduardo S. Pires**[1], **Dimas Cassimiro do N. Filho**[1,2]

[1]Universidade Federal de Campina Grande (UFCG) – Campina Grande – PB – Brasil
Departamento de Sistemas e Computação

[2]Universidade Federal do Agreste de Pernambuco (UFAPE) – Garanhuns – PE – Brasil

eleoniliamonteiro@copin.ufcg.edu.br, cesp@dsc.ufcg.edu.br
dimas.cassimiro@ufape.edu.br

Abstract. *NoSQL (Not Only SQL) databases play a crucial role, offering flexibility in managing various types of data. However, efficiently managing this data poses challenges due to the absence or outdatedness of predefined schemas. This article proposes an approach for the incremental evolution of schemas in NoSQL databases, aiming to keep the schemas updated based on newly inserted documents. Experimental results show that this approach is significantly faster than the batch approach that needs to be applied to the entire dataset (both new and old), being on average 1.93 and 2.85 times faster for book metadata and Twitter datasets, respectively.*

Resumo. *Os bancos de dados NoSQL desempenham um papel crucial, oferecendo flexibilidade na gestão de diversos tipos de dados. No entanto, a gestão eficiente desses dados apresenta desafios devido à ausência ou desatualização de esquemas pré-definidos. Este artigo propõe uma abordagem para a evolução incremental de esquemas em bancos de dados NoSQL orientados a documentos, visando manter os esquemas atualizados com base em novos documentos inseridos. Os resultados experimentais mostram que a abordagem consegue ser mais rápida do que a abordagem em lote que precisa ser aplicada em todo o conjunto de dados (novos e antigos), sendo em média 1,93 e 2,85 vezes mais rápida para conjuntos de dados de metadados de livros e Twitter, respectivamente.*

1. Introdução

Nos últimos anos, o crescimento exponencial na geração de dados impulsionou a demanda por soluções eficazes de armazenamento, processamento e análise de grandes volumes de dados. As tecnologias de Big Data desempenham um papel crucial, fornecendo *insights* valiosos para a tomada de decisões estratégicas das organizações. Entre essas tecnologias, os bancos de dados NoSQL se destacam, oferecendo flexibilidade na gestão de dados estruturados, semi-estruturados e não estruturados, sem a necessidade de um esquema explícito[1] [Hashem and Ranc 2016, Baazizi et al. 2019, Atmatzides et al. 2022].

No entanto, a ausência (ou desatualização) de um esquema pré-definido representa um desafio pois dificulta a compreensão e interpretação dos dados. Por exemplo, a escrita

[1]Um esquema explícito pressupõe que as estruturas dos dados são definidas de forma precisa e detalhada antes da inserção dos dados.

de consultas em um banco de dados sem um esquema pré-definido precisa ser confiada a desenvolvedores que participaram da criação do banco de dados e, portanto, possuam conhecimento implícito do esquema ou extraiam manualmente o esquema por meio de consulta na coleção de documentos. A existência de um esquema possibilita a validação em tempo real dos dados, torna as consultas e operações de leitura mais eficientes, facilita a integração de ferramentas e *frameworks* e simplifica a manutenção do banco de dados [Baazizi et al. 2019, Abdelhédi et al. 2022].

O problema de gerar esquemas para extrair conjuntos de dados semi-estruturados tem sido bastante estudado [Sevilla Ruiz et al. 2015, Izquierdo and Cabot 2016, Klettke et al. 2017, Baazizi et al. 2019], com destaque para a pesquisa de [Frozza et al. 2018], que propõe uma abordagem denominada *JSON Schema Discovery* (JSD) para extrair esquemas a partir de coleções de documentos JSON e JSON estendido. Embora a abordagem seja capaz de gerar esquemas, ela não suporta a evolução dinâmica do esquema à medida que novos documentos são inseridos na coleção. Para garantir que o esquema reflita com precisão o conjunto de documentos, é necessário reexecutar a abordagem considerando toda a coleção (documentos novos e antigos).

A evolução dinâmica de esquemas é essencial para evitar decisões equivocadas em diversos domínios. No comércio eletrônico, plataformas como Amazon, Lojas Americanas e eBay capturam dados de transações, avaliações de produtos, histórico de navegação e preferências dos clientes [Phillips 2016, Purnomo 2023]. Manter esquemas atualizados facilita a integração e análise de dados, permitindo que as empresas ajustem suas estratégias (e.g. estratégias de marketing, detectar fraudes, gerenciamento de estoque e personalização de recomendações de produtos) com base em informações precisas e atuais. No domínio das redes sociais, plataformas como Twitter, Facebook, VK e Instagram coletam uma vasta gama de dados, incluindo postagens, comentários, curtidas e informações demográficas dos usuários. A atualização contínua do esquema JSON é importante para a integração e análise eficaz desses dados, permitindo a identificação de tendências, padrões de comportamento e a personalização da experiência do usuário [Amorim et al. 2022, Li et al. 2023].

A solução proposta visa abordar a necessidade de evolução dinâmica de esquemas em coleções de documentos, oferecendo uma solução incremental para a evolução dos esquemas à medida que novos documentos são inseridos na coleção. Baseada na abordagem de [Frozza et al. 2018], a solução proposta parte do princípio de que já existe um esquema JSON que representa a coleção de documentos. Conforme novos documentos são inseridos na coleção, o esquema evolui para refletir os novos documentos, reduzindo o tempo e esforço necessários para a gestão de dados em cenários dinâmicos, uma vez que evita a necessidade de considerar todos os documentos da coleção. Para isso, formulou-se a seguinte questão de pesquisa: qual é o impacto da evolução incremental de esquemas em comparação com a abordagem de [Frozza et al. 2018] (em lote), em termos de eficiência?

O restante deste documento está organizado como segue: a Seção 2 apresenta a fundamentação teórica; na Seção 3, são discutidos os trabalhos relacionados; a Seção 4 detalha a abordagem incremental proposta para evolução de esquemas; na Seção 5, é apresentada a metodologia experimental; a Seção 6 discute sobre os resultados obtidos; finalmente, a Seção 7 apresenta as conclusões do trabalho e direcionamentos futuros.

2. JSON, JSON Estendido e Esquema JSON

JSON[2] é um padrão de transferência de dados que permite uma apresentação simples e concisa de estruturas de dados em formato baseado em texto. É amplamente utilizado em sistemas distribuídos e *Application Programming Interfaces (APIs)* pois permite que diferentes sistemas troquem dados de forma padronizada e sem ambiguidades. Além do JSON padrão, que oferece os tipos de dados Object, Array, String, Number, Boolean e Null, existe o JSON estendido, que inclui tipos de dados não presentes na especificação original do JSON, tais como Date, Timestamp, Binary, ObjectID, RegExp, Long, Undefined, DBRef, Code, MinKey e MaxKey.

O padrão JSON também é adotado por Sistemas de Bancos de Dados NoSQL como o MongoDB[3]. Nesse sistema, os dados são manipulados em documentos JSON e armazenados em BSON (JSON binário). Cada documento possui pares chave-valor e é considerado um objeto único em uma coleção de documentos. A flexibilidade do formato JSON permite alterações na estrutura dos documentos sem afetar o funcionamento de aplicativos que utilizam o MongoDB [Frozza et al. 2018, Atmatzides et al. 2022].

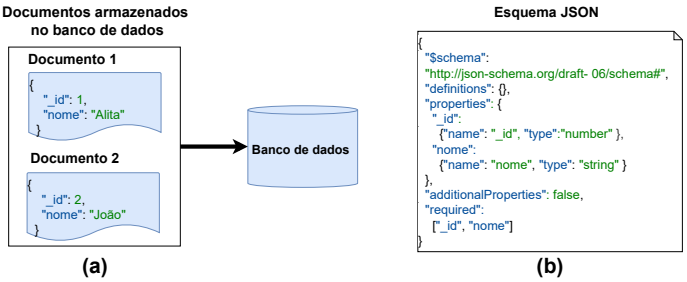


Figura 1. Exemplos de documentos JSON e Esquema JSON

Por sua vez, um esquema JSON[4] é um padrão que permite descrever e validar a estrutura, restrições e tipos de dados em documentos JSON. O padrão oferece uma linguagem declarativa escrita em forma de esquema, que descreve a estrutura do documento JSON, os tipos de dados que deve conter e as validações que devem ser aplicadas em cada atributo. Dessa forma, é possível garantir que os dados recebidos estejam no formato e na estrutura esperados. A Figura 1 apresenta dois documentos JSON e o esquema JSON que valida os mesmos. O esquema determina que os atributos *_id* e *nome* estão associados aos tipos de dados $Number$ e $String$, respectivamente, e que são obrigatórios, ou seja, devem estar presentes em todos os documentos.

3. Trabalhos Relacionados

Em [Frozza et al. 2018], é proposta uma abordagem para extrair esquemas a partir de coleções de documentos JSON e JSON estendido. A abordagem é composta por quatro etapas. Na primeira etapa, um esquema bruto é gerado para cada documento da coleção. Um esquema bruto mantém a estrutura hierárquica dos atributos, objetos aninhados e arrays (ou matrizes), preservando a organização dos dados no documento. Os valores

[2]https://www.json.org/
[3]https://www.mongodb.com/pt-br
[4]https://json-schema.org/

primitivos presentes no documento são substituídos pelos tipos de dados JSON ou JSON estendidos. Na segunda etapa, é realizada uma primeira agregação para extrair uma coleção de objetos JSON únicos. Em seguida, os atributos dos esquemas brutos obtidos na primeira agregação são ordenados em ordem alfabética (essa ordenação é realizada para reduzir os esquemas brutos únicos). Após a ordenação dos atributos, é aplicada a segunda agregação. A terceira etapa envolve a construção de uma estrutura de dados (árvore) para armazenar informações sobre a estrutura hierárquica dos esquemas brutos (Raw Schema Unified Structure - RSUS). O primeiro esquema bruto serve como base e os demais são usados para atualizar a árvore com novas informações. Na última etapa, a estrutura de árvore é transformada em um esquema JSON.

Embora a abordagem proposta por [Frozza et al. 2018] gere esquemas, ela não permite a evolução do esquema quando novos documentos são inseridos na coleção. Para obter um esquema atualizado, é necessário reaplicar a abordagem em toda a coleção (documentos novos e antigos), o que pode resultar em um aumento significativo no tempo de processamento em comparação com a solução proposta neste trabalho. Em contraste, a solução aqui proposta estende a abordagem de [Frozza et al. 2018] incluindo a evolução incremental do esquema quando novos documentos são inseridos na coleção.

Os estudos de [Sevilla Ruiz et al. 2015] e [Abdelhédi et al. 2022] concentram-se na geração de esquemas para representar conjuntos de dados, enquanto as pesquisas de [Baazizi et al. 2019], [Cánovas Izquierdo and Cabot 2013] e [Klettke et al. 2017] realizam a atualização de esquemas. É importante notar que, embora [Baazizi et al. 2019] e [Cánovas Izquierdo and Cabot 2013] realizem atualizações, estas não são conduzidas de maneira incremental. Em contraste, para evoluir o esquema existente, [Baazizi et al. 2019] propõe a criação de um novo esquema que representa os novos documentos inseridos na coleção, sendo então fundido com o esquema pré-existente. A evolução do esquema de [Cánovas Izquierdo and Cabot 2013] ocorre de forma contínua. À medida que novos documentos são inseridos na coleção, eles são utilizados para evoluir o esquema existente, garantindo sua adaptação às mudanças nos documentos.

A abordagem proposta por [Klettke et al. 2017], que visa reconstruir o histórico de mudanças no esquema de um banco de dados ao longo do tempo, é dividida em três fases. A primeira fase, extração de estrutura, seleciona os tipos de entidades a serem analisados (bancos de dados, coleções ou tabelas) e as resume em grafos. A segunda fase, derivação de operações de evolução de esquema, identifica os candidatos à evolução do esquema, destacando mudanças estruturais ao longo do tempo. A parte incremental dessa abordagem é evidenciada com marcações de tempo ordenadas e divididas em subconjuntos de entidades. Cada subconjunto é processado sequencialmente, iniciando pela extração e análise das entidades, seguido pelo preenchimento da tabela de decisão. Na fase final, resolução de ambiguidades, as operações de evolução dos esquemas são resolvidas por uma tabela de decisão, considerando dados sobre entidades, tabelas relacionadas e tipos de decisão (e.g. adicionar, renomear atributos, entre outros). No entanto, ajustes manuais são necessários para lidar com novas versões do esquema, requerendo a intervenção de um analista de dados para finalizar a produção do esquema.

A Tabela 1 apresenta um resumo dos trabalhos relacionados. Em relação aos objetivos de cada abordagem, [Baazizi et al. 2019] e [Cánovas Izquierdo and Cabot 2013] visam gerar esquemas para bancos de dados massivos e serviços da API web, respectiva-

Tabela 1. Comparação entre os trabalhos relacionados

	Formato de Entrada	Objetivo	Etapas Principais	Evolução incremental do esquema
[Baazizi et al. 2019]	JSON	Gerar esquemas para banco de dados massivos	1. Map 2. Reduce (Tipo e Rotulo)	não
[Cánovas Izquierdo and Cabot 2013]	JSON Web	Gerar esquema para os serviços da API na web	1. Pré-descoberta 2. Descoberta de serviço único 3. Descoberta multi-serviço	não
[Klettke et al. 2017]	JSON	Ferramenta para visualização e validação de esquemas	1. Extração de Estruturas 2. Derivação de operações de evolução de esquema 3. Resolução de Ambigüidades	sim
Abordagem Proposta	JSON e JSON estendido	Evolução de esquemas JSON	1. Gerar Esquemas Brutos (novos documentos) 2. Agrupamento de Esquemas brutos (novos e antigos) 3. Atualizar a Árvore RSUS 4. Gerar Esquema JSON	sim

mente, enquanto a abordagem de [Klettke et al. 2017] foi desenvolvida com o propósito de visualizar e validar esquemas. Por outro lado, nossa solução visa evoluir o esquema à medida que novos documentos são inseridos em uma coleção. Quanto ao critério de evolução incremental de esquemas, apenas [Klettke et al. 2017] e a nossa solução atendem ao critério. No entanto, a abordagem de [Klettke et al. 2017] requer a intervenção de uma analista para resolver problemas de ambiguidade, o que pode aumentar os custos associados ao processo.

4. Solução Proposta

Esta Seção apresenta a solução proposta para a evolução incremental de esquemas em conjuntos de documentos JSON. A solução parte do princípio que já existe um esquema JSON criado para representar uma coleção de documentos e que apenas os novos documentos adicionados à coleção (ou seja, documentos que não foram utilizados para a criação do esquema JSON) são considerados para a evolução do esquema. Ao considerar apenas os novos documentos, não há a necessidade de recriar o esquema do zero, o que exigiria o reprocessamento de todos os documentos (novos e antigos).

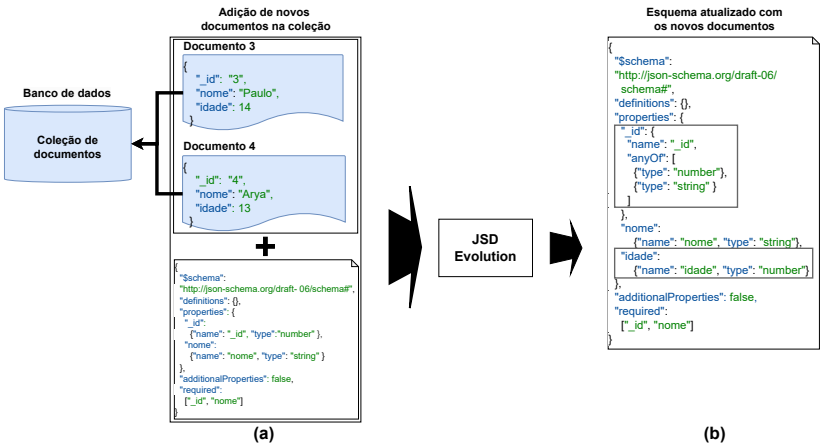


Figura 2. Inserção de novos documentos JSON e evolução do esquema JSON

Para exemplificar, considere o exemplo da Figura 2. A inserção de dois novos documentos (Documentos 3 e 4) no banco de dados (Figura 2 (a)) faz com que o esquema JSON inicial (Figura 1(b)) passe a não refletir mais o conteúdo do conjunto de documentos, uma vez que o esquema não inclui (ainda) o atributo *idade* introduzido pelos novos documentos. Para assegurar que esse esquema JSON represente o conjunto de dados

(documentos novos e antigos), nossa solução, denominada de JSD Evolution, é aplicada apenas nos novos documentos (Figura 2 (a)), resultando na evolução do esquema (Figura 2 (b)). Como resultado, o atributo *_id* pode ser do tipo $String$ ou $Number$ (na Figura 1 (b), *_id* é do tipo $Number$), e um novo atributo, *idade*, foi adicionado (sendo este atributo opcional, pois está presente apenas nos novos documentos).

4.1. Abordagem JSD Evolution

Esta seção descreve como ocorre a evolução incremental de esquemas JSON, ilustrada na Figura 3. Embora a abordagem proposta seja baseada no trabalho de [Frozza et al. 2018], a fim de torná-la incremental, foram realizadas modificações nas etapas de gerar esquemas brutos (novos documentos), agregar esquemas brutos (novos e antigos) e atualizar Árvore RSUS. Portanto, as partes em amarelo destacam as modificações realizadas, e as em cinza indicam as partes mantidas.

A abordagem proposta é dividida em quatro etapas. Na Etapa 1, são gerados os esquemas brutos, porém exclusivamente para os novos documentos inseridos no banco de dados, diferentemente da abordagem de [Frozza et al. 2018] que gera esquemas brutos para todo o conjunto de documentos (novos e antigos). Os esquemas brutos são armazenados com um identificador único, o ID do esquema JSON, e uma nova versão é atribuída. A atribuição de uma nova versão ao novo esquema bruto gerado é realizada automaticamente pela solução proposta neste artigo. Por exemplo, se for a segunda evolução do esquema JSON, a versão dos novos esquemas brutos gerados será a versão dois. A numeração das versões é importante para a Etapa 3, pois são utilizados exclusivamente os esquemas brutos com a versão mais recente para atualizar a Árvore RSUS. Na Etapa 2, é realizada a agregação dos esquemas brutos das versões anteriores e da versão atual para manter apenas esquemas brutos únicos. Na Etapa 3, o processo de geração da Árvore RSUS foi modificado para utilizar a última Árvore RSUS como base, diferentemente da abordagem de [Frozza et al. 2018] que utiliza o primeiro esquema bruto como ponto de partida. Os novos esquemas brutos são usados para atualizar a Árvore RSUS. Isso significa que, por exemplo, se for a segunda evolução do esquema JSON, apenas os esquemas brutos com versão 2 serão usados para atualizar a Árvore RSUS. A Etapa 4 é responsável por transformar a Árvore RSUS consolidada em um esquema JSON e não sofreu alterações em relação ao trabalho de [Frozza et al. 2018].

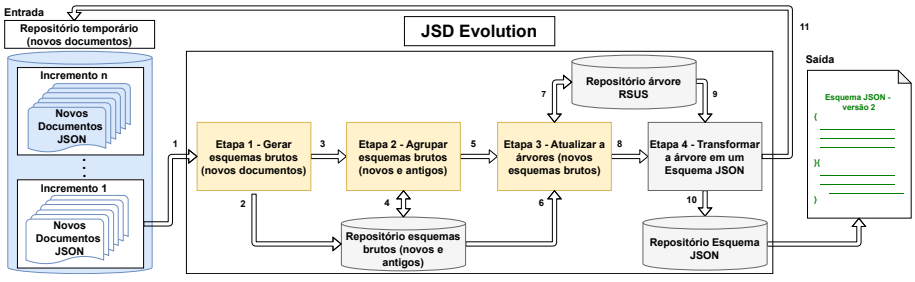


Figura 3. Fluxo da evolução incremental de esquemas

Em relação aos repositórios de dados intermediários, na abordagem JSD Evolution, foi adicionado um repositório temporário de documentos (destacado em azul) para armazenar os documentos necessários à atualização do esquema. Neste repositório, um

incremento refere-se aos novos documentos inseridos no banco de dados que serão utilizados para evoluir o esquema. O tamanho de cada incremento refere-se à quantidade de documentos contidos no incremento e é determinada por um limite pré-definido pelo usuário. Os repositórios de esquemas brutos, da árvore RSUS e do esquema JSON (destacados em cinza) foram reaproveitados da abordagem de [Frozza et al. 2018]. A seguir, o processo de evolução de esquemas é descrito em mais detalhes.

Inicialmente, considere que D é o conjunto inicial de documentos JSON; D_t é o conjunto de novos documentos; R é o conjunto contendo esquemas brutos gerados e $R_t \in R$, onde R_t é o sub conjunto de novos esquemas brutos gerados no instante t; e t é uma variável que representa um ponto específico no tempo dentro do processo de evolução dos esquemas, denotando o momento atual em que ocorrem eventos relacionados à adição de novos documentos ao conjunto de dados ou geração de novos esquemas brutos.

O processo de evolução incremental de esquemas é dividido em quatro etapas:

Etapa 1 - Gerar Esquemas Brutos (novos documentos): nesta etapa, é gerado um esquema bruto para cada novo documento $D_t \in D$ inserido no banco de dados até o instante t. Esses novos esquemas brutos são então armazenados em R, juntamente com a versão correspondente da atualização.

Etapa 2 - Agrupar Esquemas Brutos (novos e antigos): nesta etapa, é aplicada duas agregações tanto nos esquemas brutos novos quanto nos antigos, mantendo apenas esquemas únicos. Tem-se dois conjuntos para essa etapa: $R = \{r_1, r_2, ..., r_n\}$ que representa o conjunto de todos os esquemas brutos, onde r é um esquema bruto; e $A = \{a_1, a_2, ..., a_n\}$ é conjunto de atributos que pertence a um esquema bruto $r \in R$.

A primeira agregação é aplicada em R, deixando apenas esquemas brutos únicos. Em seguida, é realizada a ordenação no conjunto A de cada esquema bruto $r \in R$ resultante da primeira agregação realizada, para reduzir a quantidade de esquemas brutos distintos. Em seguida, a segunda agregação é aplicada nos esquemas brutos ordenados R, removendo informações duplicadas presentes em partes diferentes do esquema.

Etapa 3 - Atualizar a Árvore RSUS: na terceira etapa, a Árvore RSUS do esquema que está sendo atualizado é usada como base, e os novos esquemas brutos são utilizados para atualizar a RSUS. O Algoritmo 1 (Figura 4) descreve o processo de atualização da árvore RSUS com os novos esquemas brutos únicos. O algoritmo recebe como entrada a árvore original T_{Base} e o subconjunto R_t que representa os novos esquemas brutos pertencentes a R. A árvore RSUS é utilizada como base, e os novos esquemas brutos são aplicados para atualizar essa árvore. Por fim, a RSUS atualizada é retornada.

Etapa 4 - Gerar Esquema JSON: nessa etapa, ocorre a transformação da Árvore RSUS em um esquema JSON que representa o banco de dados.

$$\mathrm{E}_{final} = TransformTreeIntoJSON(T_{updated})$$

Nesta expressão, E_{final} é o esquema JSON atualizado que representa a coleção monitorada. A função $TransformTreeIntoJSON$ converte a Árvore RSUS atualizada em um esquema JSON, enquanto $T_{updated}$ é a Árvore RSUS atualizada com os novos esquemas brutos.

```
1. function UpdateRSUSWithNewRawSchemas(Rt, Tbase):
      Input: Tbase (base RSUS tree), Rt (new raw schemas)
      Output: Tupdated (updated RSUS tree)
2.    Begin
3.        if (Rt is not an array) then // Input validation
4.            Throw an error 'Rt is not an array'
5.        end if
6.        if (Tbase is undefined or does not contain valid fields) then
7.            Tbase = { fields: [], count: 0 } // Create a new empty tree
8.        end if
9.      // Keep a reference to the current root schema
10.      Tupdated = rootSchema
11.      Tupdated.count += Tbase.count
12.      // Update the existing tree with the fields from Tbase
13.      for each (matchingField in Tbase.fields) do
14.          existingField = find field in Tupdated.fields
                 where field.path equals matchingField.path
15.          if (existingField exists) then
16.              // update its count
17.              existingField.count += matchingField.count
18.          else
19.              // add it to the existing tree
20.              Tupdated.fields.push(matchingField)
21.          end if
22.      end for
23.      // Add the new raw Rt to the existing tree
24.      for each (r in Rt) do
25.          parsedID = parse r._id
26.          parsedValue = parse r.value
27.          buildRawSchema(parsedID, parsedValue, Tupdated.fields)
28.          Tupdated.count += parsedValue
29.      end for
30.      return Tupdated
31.  end function
```

Figura 4. Algoritmo de atualização da árvore RSUS

5. Avaliação Experimental

Esta seção descreve a metodologia experimental empregada para avaliar a eficiência da evolução incremental de esquemas JSON e a abordagem em lote proposta por [Frozza et al. 2018]. Ademais, foi verificada a similaridade entre os esquemas JSON gerados/atualizados pelas abordagens JSD e JSD Evolution, respectivamente. Para realizar essa verificação, foi utilizada a biblioteca DeepDiff [Foundation 2024] da linguagem Python, que considera as variações nas posições dos atributos e tipos. Os experimentos foram realizados em um computador Core i7 6700 e 32 GB de RAM.

5.1. Conjuntos de dados

Para avaliar nossa abordagem, foram utilizados três conjuntos de dados: Livros, Twitter e VK, todos obtidos a partir do repositório Kaggle. Os dois últimos também foram usados em [Baazizi et al. 2019]. O conjunto de dados Twitter[5] contém tweets relacionados às eleições russas de 2018. O conjunto de dados VK[6] engloba documentos que descrevem interações dos usuários na rede social russa durante as eleições russas de 2018. Finalmente, o conjunto de dados Livros[7] abrange documentos de metadados de livros. Um resumo sobre os conjuntos de dados pode ser visto na Tabela 2.

Tabela 2. Descrição dos conjuntos de dados

Conjunto de dados	Tamanho	Quantidade de objetos	Tamanho textual médio por documento	Quantidade de atributos obrigatórios	Quantidade de atributos opcionais	Quantidade de atributos
Twitter	11.5 GB	1.945.365	5.67 KB	885	234	alta
VK	5.6 GB	3.036.654	1.80 KB	159	97	média
Livros	82.2 GB	47.015.693	1.76 KB	32	1	baixa

[5]https://www.kaggle.com/datasets/borisch/russian-election-2018-twitter

[6]https://www.kaggle.com/datasets/borisch/russian-election-2018-vkcom-user-activity

[7]https://www.kaggle.com/datasets/opalskies/large-books-metadata-dataset-50-mill-entries

5.2. Métricas de Avaliação

A métrica usada para avaliar o desempenho das abordagens de atualização incremental de esquemas (JSD Evolution) e em lote (JSD) é o tempo de processamento. Foi registrado o tempo de execução consumido pela JSD para gerar o esquema JSON que representa o conjunto de dados (documentos novos e antigos) e o tempo de execução gasto pela abordagem de atualização incremental (JSD Evolution) para atualizar o esquema JSON existente considerando apenas os novos documentos inseridos na coleção.

Neste contexto, os tempos de execução das abordagens são usados para responder à pergunta de pesquisa: "Qual é o impacto da atualização incremental de esquemas em comparação com a abordagem em lote [Frozza et al. 2018], em termos de eficiência?"

5.3. Preparação dos Conjuntos de Dados

Para conduzir os testes da atualização incremental e em lote, foram desenvolvidas abordagens distintas na preparação dos conjuntos de dados. O objetivo foi avaliar o desempenho dessas abordagens na evolução do esquema JSON em diferentes contextos, nos quais os experimentos podem variar quanto ao conjunto de dados utilizado, ao tamanho dos incrementos (quantidade de novos documentos) e à quantidade de atributos por documento (e.g. baixa, média ou alta quantidade de atributos por documento). A seguir, são detalhadas as etapas dessas preparações.

5.3.1. Preparação dos dados para abordagem de evolução incremental

Para os testes de evolução incremental, utilizou-se uma Progressão Geométrica (PG) para dividir os dados em partições, representando os novos documentos usados para evoluir o esquema. A primeira partição foi usada para criar o esquema inicial, enquanto as partições seguintes foram utilizadas para evoluir o esquema atual. Essas partições refletem diferentes estágios de crescimento da quantidade de documentos. Por exemplo, é possível observar como o esquema inicial evolui conforme mais dados são adicionados e como essa evolução afeta o desempenho em termos de tempo de processamento. A PG permite distribuir os dados em partições de forma gradual, facilitando a avaliação do impacto de atualizações incrementais em diferentes cenários de crescimento de dados.

Para determinar o primeiro termo e a razão da PG, foi utilizado apenas números inteiros. Isso foi feito para garantir que a distribuição dos dados em partições resultasse em quantidades inteiras. O conjunto de dados do Twitter foi particionado usando uma PG, com primeiro termo de 450 e razão de 2, simulando incrementos progressivos na quantidade de documentos. Esse conjunto de dados foi dividido considerando a quantidade de documentos da coleção e os atributos por documento. A soma dos 12 primeiros termos dessa PG resulta em 1.872.500, aproximando-se do total de documentos disponíveis na coleção do Twitter, que é de 1.945.365. Além disso, essa coleção apresenta uma alta quantidade de atributos (ver Tabela 2). Para o conjunto de dados do VK, foram adotados os mesmos valores da PG usados para particionar a coleção de documentos do Twitter (primeiro termo de 450 e uma razão de 2). Esses particionamentos permitem analisar como as abordagens de evolução incremental e em lote se comportam diante de cenários com a mesma quantidade de documentos, mas com alta e média quantidade de atributos por documento nas coleções (Twitter e VK, respectivamente).

No caso do conjunto de dados Livros, foi adotada uma PG com primeiro termo de 10.000 e razão 2, refletindo o crescimento progressivo do número de documentos. Os

valores da PG foram escolhidos com base na natureza e no tamanho de cada conjunto de dados, com o objetivo de assegurar uma representação do crescimento progressivo do número de documentos em cada coleção.

5.3.2. Preparação dos dados para Abordagem em Lote

Adotou-se a estratégia de união de conjunto de dados para refletir a natureza do processamento em lotes, na qual as atualizações consideram a totalidade dos documentos acumulados até o momento (documentos novos e antigos). Essa estratégia de união de conjuntos de dados foi aplicada nas coleções usadas nos experimentos da abordagem de evolução incremental de esquemas.

Seja P_i o conjunto da partição i, tal que i varia de 1 a n, sendo n o número total de partições (partições da PG). Então, para cada partição P_i, tem-se:

- P_1 : Partição inicial contendo a_1, onde a_1 é o primeiro termo da PG (representa o primeiro conjunto de dados particionado para os testes da evolução incremental);
- P_2 : $P_1 \cup a_2$, onde a_2 é o segundo termo da PG, (representa o segundo conjunto de dados particionado para os testes da evolução incremental);
- P_{i+1} : $P_i \cup a_{i+1}$, onde a_{i+1} é o próximo termo da PG.

Cada partição representa a união dos documentos das partições anteriores, simulando o crescimento gradual e cumulativo do conjunto de dados em cenários de evolução em lote.

5.3.3. O Controlador e a Configuração de Entrada na Abordagem Proposta

O Controlador e a configuração de entrada desempenham papéis cruciais na execução dos experimentos, sendo essenciais para gerar a carga de trabalho necessária para a nossa abordagem. O Controlador monitora a inserção de novos documentos na coleção e verifica se o número de novos documentos adicionados atingiu ou excedeu o limite predefinido pelo usuário, estabelecido em termos da quantidade de novos documentos na coleção, determinando assim se uma atualização do esquema é necessária.

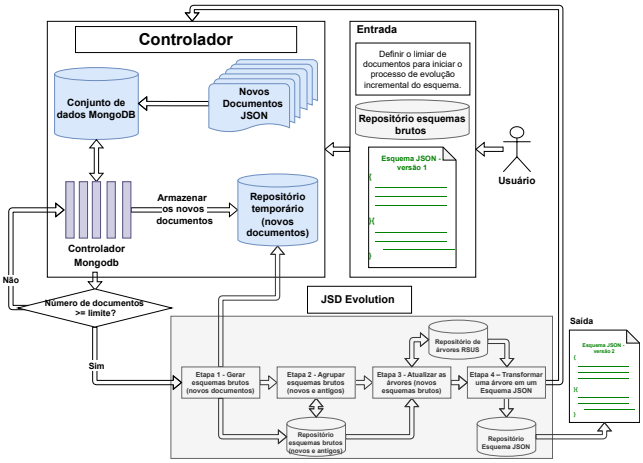


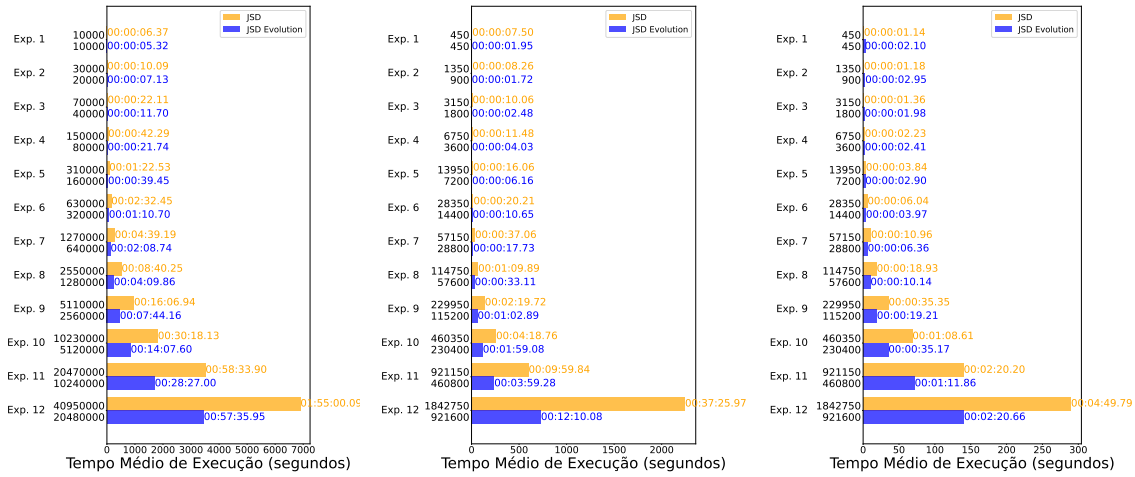
Figura 5. Controlador: usado para monitorar o banco de dados

A Figura 5 ilustra a configuração de entrada e o funcionamento do Controlador. Este processo compreende a identificação dos dados essenciais para iniciar o monitoramento e a atualização, incluindo a entrada de novos documentos, a última versão do esquema JSON, os esquemas brutos gerados nas versões anteriores e o limite de documentos

para acionar a atualização (definido pelo usuário). Os novos documentos são temporariamente armazenados em um repositório e utilizados durante o processo de evolução do esquema (JSD Evolution). O código-fonte do JSD Evolution, assim como o controlador e os scripts de testes, estão disponíveis no GitHub: https://github.com/Eleonilia/JSD-Evolution.

6. Resultados Experimentais

Foram executados 36 experimentos de desempenho, variando o conjunto de dados, o tamanho dos incrementos e a quantidade de atributos por documento, como descrito nas Seções 5.3.1 e 5.3.2. Os experimentos foram repetidos 10 vezes e considerou-se a média dos resultados obtidos. A Figura 6 ilustra o comportamento de cada abordagem nos experimentos com o conjunto de dados Livros, Twitter e VK. Os eixos x e y representam o tempo médio de execução e a quantidade de documentos utilizados nos experimentos, respectivamente. A Figura 6 (a) ilustra o desempenho das abordagens nos experimentos conduzidos no conjunto de dados de metadados de Livros. Nessa coleção, os documentos possuem poucos atributos. Observa-se que a solução JSD Evolution é, em média, 1.93 vez mais rápida do que a abordagem em lote [Frozza et al. 2018]. Para os experimentos realizados com o conjunto de dados do Twitter (Figura 6 (b)), que contém uma alta quantidade de atributos por documento, também comprova que nossa solução é mais eficiente que a abordagem em lote, sendo, em média, 2.85 vezes mais rápida.



(a) Livros **(b) Twitter** **(c) VK**

Figura 6. Comparação do Tempo Médio de Execução entre as Abordagens de Evolução Incremental (JSD Evolution) e em Lote (JSD)

Nos experimentos que envolvem uma quantidade média de atributos (Figura 6 (c)), no conjunto de documentos VK, é possível observar que os experimentos com até 6.750 documentos (exp. 1 a exp. 4), a abordagem em lote obteve um tempo de execução menor em comparação com a nossa solução, sendo em média 1.71 vez mais rápida. No entanto, à medida que a quantidade de documentos aumenta (do exp. 5 ao exp. 12), a nossa solução demonstra uma vantagem em termos de tempo de execução, conseguindo evoluir o esquema 1.77 vez mais rápido.

Além da análise de eficiência, foi realizada uma análise funcional para verificar a capacidade das abordagens JSD e JSD Evolution em identificar corretamente diversos

recursos presentes em um esquemas JSON. Foram usados 8 conjuntos de dados criados por [Latták and Koupil 2022], nomeados para refletir seus conteúdos, cada um focado em um tipo de recurso de esquema JSON, como *PrimitiveTypes, SimpleArrays, SimpleObjects, ComplexArrays, ComplexObjects, Optional, Union* e *References*. Os experimentos mostraram que ambas as abordagens produziram resultados semelhantes na geração/atualização dos esquemas JSON, diferindo apenas na posição de alguns atributos e tipos. Essa diferença pode ser atribuída ao fato de que a abordagem JSD Evolution, na etapa 3, utiliza a Árvore RSUS como base e os novos esquemas brutos gerados são usados para evoluí-la. Na etapa 3 da abordagem JSD, o primeiro esquema bruto gerado é usado como ponto de partida para a construção da Árvore RSUS, e os demais esquemas brutos (novos e antigos) são usados para atualizá-la. Ao analisar o comportamento funcional das abordagens, observou-se que, mesmo após as adaptações para permitir a evolução incremental do esquema, a abordagem JSD Evolution foi capaz de identificar todos os tipos de recurso de esquema JSON testados.

Essas diferenças de desempenho podem ser atribuídas à forma como cada abordagem lida com inserções no banco de dados. A abordagem em lote, que requer a reaplicação de todo o processo de geração de esquema JSON em todo o conjunto de documentos (documentos novos e antigos), demonstra um aumento considerável no tempo de execução à medida que a quantidade de documentos aumenta. Isso se deve ao fato de que a abordagem em lote não diferencia entre documentos novos e antigos, sendo necessário reprocessar todos os documentos a cada atualização de esquema.

Por outro lado, a solução proposta se mostra mais eficiente nesse aspecto. Ao lidar apenas com os documentos recém-inseridos, ela é capaz de evoluir o esquema de forma eficiente, resultando em tempos de execução significativamente menores, especialmente para experimentos com grande quantidade de documentos. Esses resultados destacam a vantagem da nossa solução para evolução de esquema em conjuntos de dados grandes, reduzindo o tempo de processamento ao focar nos documentos novos.

7. Conclusões e Trabalhos Futuros

Este artigo propõe uma abordagem para evoluir esquemas de forma incremental, utilizando novos documentos inseridos no banco de dados. A principal motivação é contribuir para áreas em constantes mudanças e com grande fluxo de dados (e.g. Redes Sociais e Comércio Eletrônico), mantendo os esquemas atualizados e diminuindo o tempo de execução, sem a necessidade de aplicar a abordagem em todos os documentos (novos e antigos). Com base nos resultados experimentais, é possível concluir que a solução proposta mostrou-se ser, em média, 1,93 e 2,85 vezes mais rápida que a abordagem [Frozza et al. 2018] para os conjuntos de dados de metadados de livros e Twitter, respectivamente. Para o conjunto de dados VK, a solução proposta obteve um tempo de execução menor nos experimentos 5 a 12, à medida que a quantidade de documentos aumentou.

Como trabalhos futuros, planeja-se incluir novas funcionalidades na atualização incremental de esquemas. Além de considerar a atualização do esquema ao inserir novos documentos, pretende-se incorporar a atualização de esquema quando documentos ou atributos forem removidos, atualizados ou movidos. Além disso, pretende-se aprimorar o desempenho da primeira etapa por meio do uso de processamento paralelo.

Referências

[Abdelhédi et al. 2022] Abdelhédi, F., Rajhi, H., and Zurfluh, G. (2022). Extraction process of the logical schema of a document-oriented nosql database. In *Proceedings of the 10th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2022)*, pages 61–71.

[Amorim et al. 2022] Amorim, A., Murrugarra-Llerena, N., Silva, V., de Oliveira, D., and Paes, A. (2022). Modelagem de tópicos em textos curtos: uma avaliação experimental. In *Anais do XXXVII Simpósio Brasileiro de Bancos de Dados*, pages 254–266, Porto Alegre, RS, Brasil. SBC.

[Atmatzides et al. 2022] Atmatzides, N., Bedo, M., and de Oliveira, D. (2022). Adoção de sgbds nosql em empresas brasileiras: um levantamento preliminar. In *Anais do XXXVII Simpósio Brasileiro de Bancos de Dados*, pages 385–390, Porto Alegre, RS, Brasil. SBC.

[Baazizi et al. 2019] Baazizi, M.-A., Colazzo, D., Ghelli, G., and Sartiani, C. (2019). Parametric schema inference for massive json datasets. *The VLDB Journal*, 28:497–521.

[Cánovas Izquierdo and Cabot 2013] Cánovas Izquierdo, J. L. and Cabot, J. (2013). Discovering implicit schemas in json data. In *Web Engineering: 13th International Conference, ICWE 2013, Aalborg, Denmark, July 8-12, 2013. Proceedings 13*, pages 68–83. Springer.

[Foundation 2024] Foundation, P. S. (2024). Deepdiff: Deep difference and search of any python object/data. recreate objects by adding deltas to each other. Accessed: 2024-07-26.

[Frozza et al. 2018] Frozza, A. A., dos Santos Mello, R., and da Costa, F. d. S. (2018). An approach for schema extraction of json and extended json document collections. In *IEEE International Conference on Information Reuse and Integration (IRI)*, pages 356–363.

[Hashem and Ranc 2016] Hashem, H. and Ranc, D. (2016). Evaluating nosql document oriented data model. In *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, pages 51–56.

[Izquierdo and Cabot 2016] Izquierdo, J. L. C. and Cabot, J. (2016). Jsondiscoverer: Visualizing the schema lurking behind json documents. *Knowledge-Based Systems*, 103:52–55.

[Klettke et al. 2017] Klettke, M., Awolin, H., Störl, U., Müller, D., and Scherzinger, S. (2017). Uncovering the evolution history of data lakes. In *2017 IEEE international conference on big data (Big Data)*, pages 2462–2471. IEEE.

[Latták and Koupil 2022] Latták, I. V. and Koupil, P. (2022). A comparative analysis of json schema inference algorithms. In *Proceedings of the 17th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2022)*, pages 379–386.

[Li et al. 2023] Li, X., Sun, L., Ling, M., and Peng, Y. (2023). A survey of graph neural network based recommendation in social networks. *Neurocomputing*, 549:126441.

[Phillips 2016] Phillips, J. (2016). *Ecommerce analytics: analyze and improve the impact of your digital strategy*. FT Press.

[Purnomo 2023] Purnomo, Y. J. (2023). Digital marketing strategy to increase sales conversion on e-commerce platforms. *Journal of Contemporary Administration and Management (ADMAN)*, 1(2):54–62.

[Sevilla Ruiz et al. 2015] Sevilla Ruiz, D., Morales, S. F., and García Molina, J. (2015). Inferring versioned schemas from nosql databases and its applications. In *Conceptual Modeling: 34th International Conference, ER 2015, Stockholm, Sweden, October 19-22, 2015, Proceedings 34*, pages 467–480. Springer.