Abordagem de migração de bases relacionais para bases orientadas a documentos apoiada por LLM

Luan Felipe Marmentini[1], **Evandro Miguel Kuszera**[1]

[1]Universidade Tecnológica Federal do Paraná (UTFPR)
Dois Vizinhos – PR – Brazil

luanmarmentini@alunos.utfpr.edu.br, evandrokuszera@utfpr.edu.br

Abstract. *Relational databases are widely used to store data from various applications. However, they have limitations in scenarios involving high scalability and semi-structured data, creating opportunities for the use of NoSQL databases. In such cases, approaches for migrating data between relational and non-relational databases are crucial. This paper proposes an approach that generates data migration routines, using LLMs (Large Language Models) to analyze metadata from the relational database and generate a target NoSQL schema based on a set of queries provided by the user. Experiments conducted to migrate a PostgreSQL database to MongoDB demonstrated the feasibility of the approach.*

Resumo. *Banco de dados relacionais são largamente utilizados para armazenar dados de aplicações diversas. No entanto, apresentam limitações em cenários com alta escalabilidade e com dados semi-estruturados, abrindo espaço para o uso de bancos NoSQL. Nestes cenários, abordagens para migrar dados entre bases relacionais e não relacionais são cruciais. Este artigo propõem uma abordagem que gera rotinas de migração de dados, usando LLMs (Large Language Models) para analisar metadados da base relacional e gerar esquema de destino NoSQL com base em um conjunto de consultas fornecido pelo usuário. Experimentos realizados para migrar uma base PostgreSQL para MongoDB mostraram a viabilidade da abordagem.*

1. Introdução

Banco de dados relacionais são largamente utilizados para armazenar dados de aplicações diversas. No entanto, apresentam limitações em cenários com alta escalabilidade e com dados semiestruturados, abrindo espaço para o uso de bancos NoSQL [Stonebraker et al. 2007]. Bancos de dados NoSQL (sigla do inglês *Not only SQL*) [Sadalage and Fowler 2013] diferem dos bancos relacionais em termos de arquitetura, modelo de dados e linguagem de consulta. Os bancos de dados NoSQL procuram diminuir o tempo de execução de consultas e inserções, possuindo características que proporcionam maior flexibilidade para o projeto mas, por outro lado, transferem o gerenciamento do esquema de dados e consistência de dados para a aplicação.

Sendo assim, muitos projetos optam atualmente por uma solução híbrida, utilizando bancos relacionais e não relacionais. Nesses projetos, converter e migrar os dados entre diferentes bancos de dados acaba se tornando uma necessidade. Há diferentes abordagens para converter e migrar os dados de um banco de dados relacional para um

NoSQL [Santos and Costa 2016, Lee and Zheng 2015, Vajk et al. 2013, Zhao et al. 2014, Serrano et al. 2015, Karnitis and Arnicans 2015, Jia et al. 2016, Mior et al. 2016]. Cabe ao desenvolvedor estudar, analisar e codificar a melhor solução no contexto específico, o que pode ser bastante tedioso e propenso a erros de implementação.

Neste artigo é proposta uma abordagem para migrar bases relacionais para NoSQL orientado a documentos. A partir de metadados da base relacional e um conjunto de consultas de leitura (carga de trabalho), a abordagem usa LLM (GPT-o4-mini) para analisar o contexto, sugerir esquemas de dados e executar a migração de dados. A abordagem gera código Java para realizar a migração de dados entre as bases relacional (PostgreSQL) e NoSQL (MongoDB). Se desejado, é possível usar as classes Java após a migração de dados para permitir que aplicações existentes acessem os dados no banco NoSQL. Foram executados experimentos para avaliar a abordagem, com foco em dois cenários: (*i*) geração de esquemas com baixa redundância de dados; (*ii*) geração de esquemas que priorizem desempenho em operações de leitura, com base na carga de trabalho informada. Os resultados mostram que a abordagem é viável e útil para auxiliar desenvolvedores em processo de migração entre bases de dados.

O restante deste artigo está organizado da seguinte forma. A próxima seção apresenta trabalhos relacionados. Na Seção 3 é apresentada abordagem proposta para converter e migrar bases relacionais para não relacionais. Experimentos e principais resultados são discutidos na Seção 4. A Seção 5 conclui o estudo e apresenta direções para trabalhos futuros.

2. Fundamentação Teórica e Trabalhos Relacionados

Bancos de dados relacionais são baseados em tabelas, linhas e colunas, na qual cada linha representa uma instância de uma entidade da aplicação e cada coluna um de seus atributos. De forma diferente, bancos de dados NoSQL [Sadalage and Fowler 2013] não seguem o modelo relacional, sendo geralmente classificados de acordo com sua arquitetura, modelo de dados e linguagem de consulta. Os principais modelos de dados são: orientado a colunas, chave-valor, grafo e orientado a documentos[Lóscio et al. 2011].

Bancos de dados orientados a colunas se diferenciam dos relacionais armazenando os dados fisicamente em colunas ao invés de linhas. Outra característica é a indexação tripla das entidades (linha, coluna e *timestamp*), possibilitando o versionamento dos dados persistidos. Já os bancos de dados orientados a grafos estruturam os dados em grafos, de forma que os registros são representados pelos nós e os relacionamentos pelas arestas. Este tipo de banco de dado é eficiente na realização de consultas que envolvem relacionamento entre entidades. Nos bancos de dados chave-valor, dados simples são persistidos e associados a uma chave única, similar a uma tabela *hash*. Os bancos de dados orientados a documentos se caracterizam por persistir os dados em formato JSON (*Javascript Object Notation*), permitindo armazenar dados estruturados e semiestruturados, possibilitando representar os dados de múltiplas formas, de forma normalizada e/ou desnormalizada [Hamouda et al. 2023]. Neste trabalho será utilizado o banco de dados MongoDB[1] como banco de dados de destino.

Os grandes modelos de linguagem (LLMs do inglês, *Large Language Models*), como ChatGPT-4 e Gemini são ferramentas poderosas para processamento de linguagem

[1]https://www.mongodb.com/

natural (PLN). Sua capacidade de generalizar a partir de grandes conjuntos de dados permite seu uso em diversos cenários, incluindo o processo de migração entre bases de dados [Minaee et al. 2025]. Essas características permitem seu uso no processo de migrar dados entre bancos de dados relacionais e não relacionais, por meio de técnicas de *prompt engineering*, para extrair metadados do banco relacional, analisar consultas existentes (carga de trabalho) e sugerir esquemas de dados que priorizem determinados padrões de acesso, fornecendo subsídios para a migração de dados.

Foram propostas diferentes soluções para converter bases relacionais para NoSQL. Essas soluções geralmente aplicam técnicas de desnormalização com base na análise de dependências entre tabelas e/ou padrão de acesso aos dados. Algumas delas executam algoritmos de conversão automáticos e não suportam customizações do processo de conversão, como seleção de tabelas, campos ou instâncias [Santos and Costa 2016, Lee and Zheng 2015, Vajk et al. 2013, Zhao et al. 2014, Serrano et al. 2015, Stanescu et al. 2016, Zhao et al. 2014, Freitas et al. 2016]. Outras soluções permitem ao usuário customizar a conversão das entidades, mas são soluções específicas que focam em apenas um modelo NoSQL [Karnitis and Arnicans 2015, Jia et al. 2016]. Há também soluções que usam o padrão de acesso da aplicação (consultas) para gerar um conjunto de esquemas NoSQL candidatos [Mior et al. 2016, Xiang Li et al. 2014], comparando qual deles é mais adequado de acordo com o cenário alvo [Kuszera et al. 2022].

De forma diferente, neste artigo é apresentada uma abordagem que faz uso de LLMs para auxiliar no processo de geração e análise de esquemas candidatos, para posterior uso no processo de migração de dados de bases relacionais para bases NoSQL orientadas a documentos.

3. Abordagem de Migração

Esta seção apresenta a abordagem de migração de dados entre bases relacionais para bases NoSQL orientadas a documentos, por meio do uso de LLMs para auxiliar no processo de análise do esquema de origem (relacional) e o conjunto de consultas existentes (carga de trabalho), para depois sugerir esquemas de destino e geração de código para executar a migração dos dados de origem para o destino.

3.1. Cenário de Migração de Dados

Considere um banco de dados relacional PostgreSQL que armazena dados de voos entre aeroportos de origem e destino, conforme mostra a Figura 1. O objetivo é migrar os dados para uma base de dados NoSQL orientada a documentos. Um dos desafios é decidir como os dados da base relacional serão estruturados na base de dados de destino, visto que é possível armazenar os dados de forma estruturada ou semiestruturada, de forma normalizada ou desnormalizada. Dentro deste contexto, duas abordagens são consideradas: (*i*) Converter e migrar os dados de forma que o modelo final vise melhor desempenho na recuperação dos dados; e (*ii*) converter e migrar os dados de forma que o modelo final vise menor redundância. O cenário acima é utilizado para apresentar a abordagem de migração de dados.

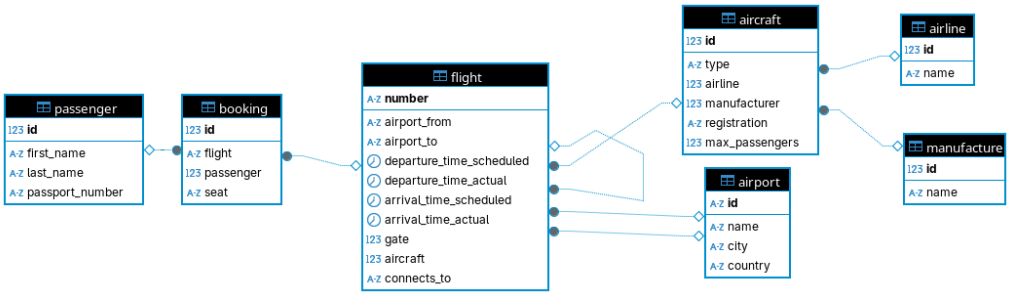

Figura 1. Esquema lógico do banco de dados relacional utilizado nos experimentos.

3.2. Arquitetura da Abordagem

A arquitetura da abordagem é estruturada como uma *pipeline* composta por 6 módulos, conforme visto na Figura 2A. Cada um dos módulos recebe um parâmetro de entrada e retorna um parâmetro de saída. Na Figura 2B são apresentadas as etapas da migração (descritas na Seção 3.3), em que cada etapa é executada por um dos módulos descritos a seguir:

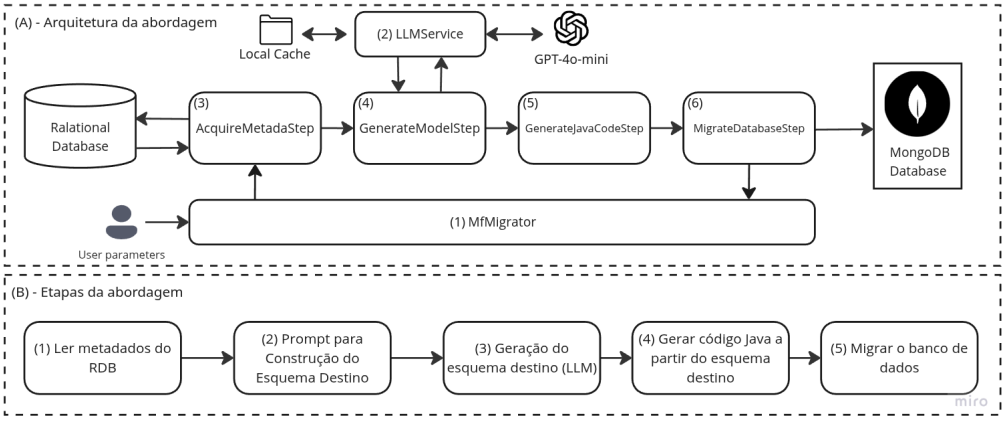

Figura 2. Arquitetura e Etapas da Abordagem.

1. **MfMigrator**: Responsável por receber os parâmetros iniciais e coordenar a execução dos demais módulos;
2. **LLMService**: Abstrai a execução dos *prompts*, enviando requisições à API da *OpenAI* ou utilizando o *cache* local quando possível.
3. **AcquireMetadaStep**: Realiza a extração dos metadados do banco de dados de origem (relacional);
4. **GenerateModelStep**: Responsável pela elaboração do *prompt* e processamento pelo modelo de linguagem, utilizando o módulo auxiliar *LLMService*;
5. **GenerateJavaCodeStep**: Cria códigos Java a partir do modelo gerado pela LLM;
6. **MigrateDatabaseStep**: Compila os códigos Java, bem como converte e persiste os dados do banco de dados relacional no banco de dados MongoDB.

A abordagem foi desenvolvida utilizando a linguagem Java, com o apoio do JDBC para acesso ao banco de dados relacional, *Spring Boot MongoDB* para a persistência dos

dados no banco de dados NoSQL MongoDB, *LangChain4j* para a comunicação entre a ferramenta e a API da *OpenIA* e *JavaParser* para a geração do código Java.

O módulo *MfMigrator* é responsável por coordenar a execução da abordagem e requer os parâmetros de entrada a seguir. **Credenciais de acesso dos bancos de dados:** informações como *host*, usuário e senha dos bancos de dados de origem e de destino. **Chave da API da LLM:** para executar os *prompts* da abordagem via requisições para a API da LLM. **Carga de trabalho da base relacional:** conjunto de consultas de leitura e frequência de execução, para que a LLM possa considerar o padrão de acesso ao sugerir um esquema de dados de destino.

3.3. Etapas da migração

A Figura 2B apresenta as etapas executadas pela abordagem de migração, sendo: (1) Ler metadados do banco de dados relacional, (2) *Prompt* para Construção do Esquema Destino, (3) Geração do esquema destino por meio da LLM, (4) Gerar código Java a partir do esquema destino e (5) Migração de dados. A seguir elas são descritas em maiores detalhes.

3.3.1. Ler metadados do banco de dados relacional

De posse das credenciais de acesso, a abordagem extrai metadados da base relacional por meio de uma conexão JDBC, como tabelas, relacionamentos, chaves primárias e estrangeiras. Após sua extração, os metadados são armazenados em uma estrutura de dados na memória. Na sequência, são executadas consultas SQL adicionais para identificar as cardinalidades mínimas, máximas e médias de cada relacionamento entre tabelas identificado no passo anterior. Esses dados são serializados em formato JSON. Esta etapa é executada pelo módulo *AcquireMetadataStep*.

3.3.2. Prompt para Construção do Esquema Destino

Esta etapa define a estrutura do *prompt* que será enviado ao LLM *gpt-o4-mini* com instruções para gerar um esquema de dados destino (MongoDB). O *prompt* usa as informações da etapa anterior (metadados e consultas SQL). O *template* utilizado para criar o *prompt* segue a estrutura abaixo:

1. **Contextualização**: contextualização sobre a tarefa, indicando que um banco de dados relacional deve ser migrado para o MongoDB e um esquema de dados deve ser gerado conforme as informações descritas a seguir;
2. **Metadados**: metadados extraídos na etapa anterior em formato SQL;
3. **Cardinalidades**: as cardinalidades anteriormente extraídas em formato JSON;
4. **Carga de trabalho da base relacional**: consultas SQL informadas pelo usuário e sua frequência de execução. Atualmente elas são fornecidas pelo usuário, mas pretende-se extrair essas informações diretamente da base de dados relacional.
5. **Esquema de Dados**: conforme a preferência do usuário, indica se o modelo gerado deve priorizar o desempenho ou menor redundância de dados.
6. **Exemplo do formato de saída esperado**: informações importantes que devem ser seguidas (estratégia *one-shot* [Brown et al. 2020]).

O *prompt* construído pela abordagem instrui a LLM a analisar os metadados do banco de dados relacional, incluindo as cardinalidades dos relacionamentos, para propor um esquema para o banco de dados destino baseado no padrão de acesso informado (*workload*). Duas orientações são enviadas via *prompt*: (*i*) "*Se duas entidades são frequentemente acessadas em conjunto, aninhe os documentos*" e (*ii*) "*Se o relacionamento tem alta cardinalidade ou os dados são atualizados com frequência ou compartilhados entre muitos documentos, utilizar referência nos relacionamentos*". Não foram definidas regras ou heurísticas complexas para a modelagem dos dados, sendo a estrutura final do esquema proveniente das decisões geradas pela LLM com base em seu conhecimento prévio. O *prompt* gerado para o cenário de exemplo pode ser visualizado na *wiki* do projeto[2].

O esquema gerado pela LLM é utilizado para gerar o código de migração de dados entre a base relacional e base orientada a documento.

3.3.3. Geração do esquema destino por meio da LLM

Nesta etapa o *prompt* é enviado para a LLM. Como resultado é gerada uma representação do esquema de dados destino baseada no formato JSON Schema[3], enriquecida com propriedades adicionais (*table*, *column*, *reference*, *docReference*, *isId* e *referenceTo*). Esses atributos adicionais são usados pela abordagem para buscar os dados da base relacional no momento da migração. Sendo assim, o JSON Schema atua como modelo de dados de destino proposto pela LLM e como um esquema de integração entre as duas bases.

```
{
    "type": "object",
    "title": "Booking",
    "properties": {
      "id": { "type": "string",  "isId": true,
        "column": "id", "table": "booking"},
      "seat": {
        "type": "string", "column": "seat", "table": "booking"
      },
      "flight": {
        "type": "string", "column": "flight",
        "table": "booking", "reference": true,
        "docReferenceTo": "Flight",
        "referenceTo": { "targetTable": "flight",  "targetColumn": "number" }
      },
      "passenger": { "type": "object",
        "column": "passenger", "table": "booking", "properties": {...},
        "referenceTo": {"targetTable": "passenger", "targetColumn": "id"} }
    }
```

Figura 3. Definição do documento *booking* gerado pela *LLM*.

A Figura 3 representa a estrutura do documento *Booking*. Nela é possível observar o uso das propriedades citadas anteriormente e que não são descritas na especificação do JSON Schema. A seguir maiores detalhes:

1. Os campos *table* e *column* representam a tabela e coluna existente na base relacional de origem. Por exemplo, no documento *Booking* a propriedade *seat* representa a coluna *seat* da tabela *booking* do banco relacional;

[2]https://github.com/EnityBlackHawk/mfcore/wiki/Prompt/
[3]https://json-schema.org/

2. O campo *referenceTo* é usado para indicar a tabela e coluna do banco relacional que uma propriedade referencia. No exemplo da Figura 3, na propriedade *passenger*, a coluna mapeada referencia a coluna *id* da tabela *passenger* do banco relacional (linha 18);
3. Os campos *reference* e *docReferenceTo* são usados para indicar o uso do relacionamento por referências entre documentos no banco NoSQL. O primeiro é uma *flag* que indica se a propriedade é uma referência e o segundo indica a qual documento essa propriedade referencia. Na Figura 3, a propriedade *flight* referencia documentos da coleção *flight* (linhas 12 e 13).

Ao final desta etapa, o usuário pode visualizar o esquema sugerido pela LLM, bem como realizar modificações. Contudo, atualmente não há uma validação automática do esquema para garantir sua conformidade com o padrão JSON Schema ou com os requisitos do processo de migração, sendo responsabilidade do usuário assegurar que, após a sua edição, o esquema final esteja correto antes de prosseguir para a próxima etapa.

3.3.4. Gerar código Java a partir do esquema destino

A ferramenta utiliza um algoritmo que transforma o JSON Schema gerado pela *LLM* em código-fonte *Java* executável. A Figura 4 mostra uma classe gerada pela ferramenta utilizando o *JSON* presente na Figura 3. Essa classe gerada representa uma coleção do banco de dados destino (MongoDB). Ela será utilizada para converter e persistir os dados entre a base relacional e orientado a documentos.

```
1  @Document()
2  @lombok.Data()
3  public class Booking {
4
5      @org.springframework.data.annotation.Id()
6      @FromRDB(type = "STRING", typeClass = java.lang.String.class, column = "id", table =
           "booking", isReference = false, isAbstract = false, projection = "*")
7      private java.lang.String id;
8
9      @FromRDB(type = "STRING", typeClass = java.lang.String.class, column = "seat", table
           = "booking", isReference = false, isAbstract = false, projection = "*")
10     private java.lang.String seat;
11
12     @org.springframework.data.mongodb.core.mapping.DBRef()
13     @FromRDB(type = "OBJECT", typeClass = Flight.class, column = "flight", table = "
           booking", isReference = true, isAbstract = false, projection = "*", targetTable
           = "flight", targetColumn = "number")
14     private Flight flight;
15
16     @FromRDB(type = "OBJECT", typeClass = Passenger.class, column = "passenger", table =
           "booking", isReference = true, isAbstract = false, projection = "*",
           targetTable = "passenger", targetColumn = "id")
17     private Passenger passenger;
18 }
```

Figura 4. Exemplo de classe Java gerada pela ferramenta.

A partir dos dados presentes no JSON Schema foram definidas anotações customizadas para facilitar a identificação das colunas que serão mapeadas para cada propriedade dentro de uma classe. Essas informações são utilizadas na realização das consultas realizadas ao banco de dados de origem. Essas anotações podem ser observadas nas linhas 6,

9, 13 e 16 da Figura 4. Ao final dessa etapa, o código-fonte Java é persistido na memória principal em formato *string* e enviado para a próxima etapa.

3.3.5. Migração de dados

Antes de iniciar o processo de migração o código-fonte gerado pela etapa anterior é compilado utilizando a API *JavaCompiler*. As classes resultantes da compilação são armazenadas em memória.

Na sequência, o processo de migração é disparado pela abordagem. A partir das classes Java compiladas são realizadas consultas sobre a base de dados relacional, resultando na carga de objetos em memória. Esses objetos são convertidos em objetos das classes geradas. O pseudocódigo abaixo exemplifica esse processo:

```
1    objs_para_persistir = []
2    for classe in classes_geradas:
3        rows[] = consultar_rdb("SELECT * FROM ${classe.name}")
4        objs[] = converter_rows_em_objs(rows, classe)
5        objs_para_persistir.inserir(objs)
6    end
```

Caso a classe tenha relacionamentos com outras classes, pode ser necessário disparar consultas secundárias para outras tabelas da base relacional. Essas consultas são geradas com base nas anotações customizadas adicionadas pela abordagem, isso pode ser observado nas linhas 9 a 11 do pseudocódigo a seguir:

```
1    objs[] converter_rows_em_objs(rows, classe):
2        objs = []
3        for row in rows:
4            obj = classe.instanciar()
5            for field in classe.fields:
6                targetTable = field.annotation.targetTable
7                targetColumn = field.annotation.targetColumn
8                if targetTable != null:
9                    depRows[] = consultar_rdb("SELECT * FROM ${targetTable} WHERE ${
                         targetColumn} = ${row[field.annotation.column]}")
10                   depObjs[] = converter_rows_em_objs(depRows, field.tipo)
11                   field.definirValor(depObjs[0], obj)
12                   continue
13               end
14               field.definirValor( row[field.annotation.column], obj )
15           end
16           obj.inserir(obj)
17       end
18       return objs
19   end
```

Por fim, a abordagem persiste os dados na base destino, convertendo os dados para o formato JSON e finalizando o processo de migração. Vale ressaltar que não há a necessidade de que as coleções sejam persistidas em uma determinada ordem, uma vez que o *MongoDB* não possui, por padrão, a verificação da integridade dos documentos.

4. Experimentos e Resultados

Para avaliar a abordagem proposta neste trabalho foram conduzidos experimentos de migração de dados de um banco de dados relacional PostgreSQL (origem) para um banco de dados NoSQL MongoDB (destino), com adaptação do esquema de destino conforme

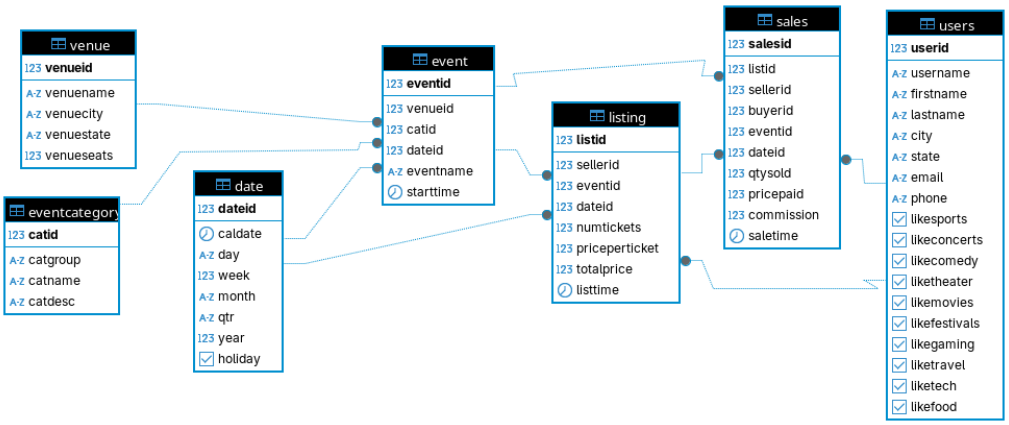

Figura 5. Esquema lógico do banco de dados relacional de origem utilizado nos experimentos.

os parâmetros de entrada definidos pela abordagem. O banco de dados de destino é composto por sete tabelas e aproximadamente 420.000 registros, representando a estrutura de um sistema de venda de ingressos. A Figura 5 apresenta o esquema lógico desse banco.

Três cenários de teste foram elaborados, com diferentes requisitos em termos de esquema de dados no banco de destino: (1) Esquema de dados normalizados; (2) Esquema de dados desnormalizados; (3) Esquema de dados conforme o *workload* informado pelo usuário. Os cenários são discutidos nas próximas seções. O *workload* consiste em 6 consultas SQL sobre a base de dados relacional de origem, representando consultas geralmente executadas pelo usuário. Essas consultas são usadas no processo de criação de esquemas na base destino e também para avaliar o desempenho em operações de leitura após migração dos dados. As consultas são apresentadas abaixo:

```sql
-- consulta 1
SELECT e.eventname, e.starttime, v.venuecity, v.venuestate
FROM event e
JOIN venue v ON e.venueid = v.venueid
WHERE e.starttime > '2005-01-01 00:00' AND v.venuecity = 'San Francisco';

-- consulta 2
SELECT ec.catname, e.eventname, e.starttime
FROM event e
JOIN eventcategory ec ON e.catid = ec.catid
WHERE ec.catname = 'Musicals';

-- consulta 3
SELECT l.listid, u.username, l.numtickets, l.priceperticket, l.listtime
FROM listing l
JOIN users u ON l.sellerid = u.userid
WHERE l.eventid = 123
ORDER BY l.priceperticket ASC;

-- consulta 4
SELECT d.month, d.year, SUM(s.qtysold) AS total_tickets, SUM(s.pricepaid) AS revenue
FROM sales s
JOIN date d ON s.dateid = d.dateid
GROUP BY d.month, d.year
ORDER BY d.year DESC, d.month DESC;

-- consulta 5
SELECT DISTINCT e.eventname, e.starttime, ec.catname
FROM users u
JOIN listing l ON u.userid = l.sellerid
```

```
    JOIN event e ON l.eventid = e.eventid
    JOIN eventcategory ec ON e.catid = ec.catid
    WHERE u.userid = 1
      AND (
        (u.likesports = true AND ec.catgroup = 'Sports') OR
        (u.likeconcerts = true AND ec.catgroup = 'Concerts') OR
        (u.liketheater = true AND ec.catgroup = 'Shows')
      )
      AND e.starttime > '2005-01-01 00:00';

    -- consulta 6
    SELECT e.eventname, SUM(s.qtysold) AS total_tickets,
           SUM(s.pricepaid) AS total_revenue,
           SUM(s.pricepaid) - SUM(s.commission) AS net_revenue
    FROM sales s
    JOIN event e ON s.eventid = e.eventid
    GROUP BY e.eventname
    ORDER BY net_revenue DESC;
```

Listing 1. Consultas que representam o *workload*.

4.1. Cenário de Teste 1

O primeiro cenário de teste visa o uso de esquema normalizado ao final da migração, estruturando os dados na base de destino mantendo a mesma estrutura da base de origem, com as entidades e relacionamentos originais. A LLM é instruída a manter o esquema similar ao relacional, ou seja, sem utilizar aninhamento entre os documentos.

4.2. Cenário de Teste 2

O segundo cenário de teste visa um esquema de dados desnormalizado, com documentos aninhados. Essa estratégia de estruturação dos dados é muito utilizada quando um grupo de informações são frequentemente acessadas em conjunto, eliminando a necessidade de operações como o *lookup* no MongoDB [Hamouda et al. 2023]. Neste caso a LLM é instruída a usar aninhamento de documentos.

4.3. Cenário de Teste 3

O terceiro cenário avalia a capacidade da LLM de elaborar um modelo adaptado às consultas SQL informadas, porém priorizando o desempenho em operações de leitura. Sendo assim, entidades que constantemente são acessadas em conjunto devem estar aninhadas. De forma diferente, entidades com grande probabilidade de sofrerem alterações e com consultas esporádicas, devem usar relacionamentos por referência.

4.4. Resultados

Após a execução dos três cenários de testes descritos anteriormente, três bancos de dados MongoDB foram criados, todos seguindo as especificações da LLM e com os mesmos dados presentes no banco de dados de origem.

A Figura 6 exibe os esquemas de dados gerados pela abordagem para os cenários 2 e 3. O esquema do cenário 1 (Esquema 1) não é apresentado na Figura 6, pois ele segue a mesma estrutura do banco relacional de origem (sem aninhamentos entre documentos, apenas uso de referências). Cada esquema é representado por meio de um conjunto de grafos. Vértices representam documentos e arestas relacionamentos entre documentos. Vértices raiz representam coleções de documentos e os demais vértices os respectivos

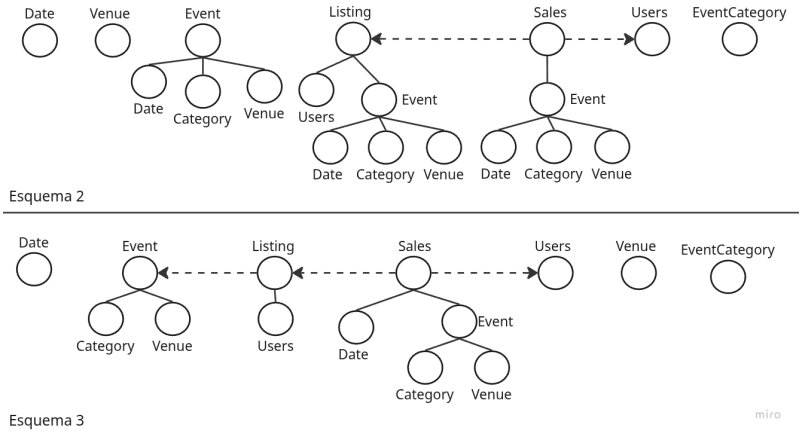

Figura 6. Grafos representando os esquemas gerados.

documentos aninhados (ou embutidos). Há também arestas segmentadas, que representam relacionamentos do tipo referência entre documentos.

É possível observar que a LLM relaciona todas as tabelas da base de dados de origem para criar os esquemas de dados para o banco de dados MongoDB. Experimentos foram realizados para avaliar os esquemas gerados em relação a carga de trabalho informada pelo usuário. As consultas SQL da Listagem 1 foram convertidas para a linguagem do MongoDB, a fim de verificar o resultado retornado e também o tempo de resposta para cada banco de dados gerado.

Cada consulta foi executada 30 vezes para calcular o tempo de execução médio, descartando os tempos máximo e mínimo. A Tabela 1 mostra a relação entre os esquemas ($S1 - S3$) e o tempo médio das execuções das consultas ($Q1 - Q6$) em milissegundos (ms), além da soma das médias das execuções e o tamanho final (em megabytes) do banco de dados.

Schema	$Q1$	$Q2$	$Q3$	$Q4$	$Q5$	$Q6$	*Sum*	*Size* (MB)
S1	579.75	588.03	84.90	382.52	**85.29**	6228.55	7,949.05	**26.08**
S2	**7.88**	**9.33**	108.80	382.05	213.64	**136.49**	858.20	97.95
S3	10.99	10.21	**88.32**	**125.24**	93.74	137.76	**466.26**	67.99

Tabela 1. Tempo de execução das consultas (Q) em milissegundos e tamanho dos esquemas (S) gerados.

- **Esquema 1**: teve o maior tempo de execução. Isso se deve a normalização dos dados, sendo necessário o uso de operações de *lookup* para responder a maioria das consultas do experimento. Ademais, não foram criados índices sobre as coleções de documentos, onerando ainda mais as operações de *lookup* sobre o esquema normalizado.
- **Esquema 2**: teve o segundo maior tempo de execução. Nesse esquema os dados estão desnormalizados, mas não correspondem ao padrão de acesso do conjunto de consultas, necessitando realizar transformações para retornar o resultado ao usuário.
- **Esquema 3**: teve o menor tempo de resposta. Esse esquema usa desnormalização e as coleções de documentos são mais próximas ao padrão de acesso do conjunto

de consultas, não necessitando executar transformações para retornar o resultado ao usuário.

Os resultados indicam que o padrão de acesso deve ser levado em consideração ao criar esquemas que preconizam melhor desempenho em operações de leitura. Em relação ao tamanho das bases de dados, esquemas desnormalizados ocupam maior espaço, em função da redundância de dados. Por exemplo, o Esquema 1, com dados normalizados, é aproximadamente 3 vezes menor que o Esquema 2, que preconiza a desnormalização de dados.

O uso da LLM para analisar e propor esquemas de dados mostrou ser uma ferramenta de grande utilidade, sugerindo esquemas que apresentam melhor desempenho em relação ao conjunto de consultas de entrada. Em contra partida, mostrou capacidade de gerar modelo de dados que preconizam desnormalização, alcançando bom desempenho em operações de leitura, mesmo sem usar o conjunto de consultas de entrada. Vale ressaltar que em alguns casos foi necessário realizar ajustes manuais no JSON Schema gerado pela LLM. Nesses casos, as propriedades customizadas não eram definidas corretamente pela LLM ou os valores atribuídas a elas eram inconsistentes. Esses problemas serão investigados no futuro e podem ser minimizados com o aprimoramento do *prompt* ou não delegar esta etapa da a LLM.

5. Considerações Finais

A abordagem descrita neste trabalho migrou os dados de um banco de dados origem (relacional) para um banco de dados destino (orientado a documentos) utilizando uma LLM para realizar a modelagem dos dados conforme as preferências do usuário. Três cenários são usados para avaliar abordagem, considerando esquemas de dados normalizados, desnornalizados e baseados no *workload* do banco relacional de origem.

Apesar de algumas vezes ser necessário realizar pequenos ajustes no JSON Schema gerado pela LLM (devido à alta complexidade da representação), a migração foi realizada corretamente, com todos os dados do banco de dados de origem sendo migrados para o banco de dados de destino. Além disso, a LLM é capaz de adaptar o esquema conforme as consultas fornecidas, mesmo que a variação entre priorizar o desempenho ou a redução da redundância de dados não tenha mostrado uma mudança significativa no tamanho, ou na disposição dos documentos.

Como trabalho futuro pretende-se aprimorar os *prompts* por meio do uso de técnicas como *few-shots* ou *chain of thought* para não somente evitar os erros de representação anteriormente mencionados, mas também para melhorar a capacidade de otimização dos modelos gerados pela LLM. Também se faz necessário realizar avaliações com bases de dados maiores e complexas. Por fim, também pretende-se desenvolver uma ferramenta visual para auxiliar o usuário no processo de definição de esquemas de dados e migração.

Agradecimentos

Este trabalho contou com o apoio do CNPq, por meio da bolsa de Iniciação Científica concedida ao estudante.

Referências

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners.

Freitas, M. C. d., Souza, D. Y., and Salgado, A. C. (2016). Conceptual Mappings to Convert Relational into NoSQL Databases. In *Proceedings of the 18th International Conference on Enterprise Information Systems*, ICEIS 2016, page 174–181, Setubal, PRT. SCITEPRESS - Science and Technology Publications, Lda.

Hamouda, S., Anbar, M., and Elejla, O. E. (2023). Normalization and denormalization for a document-oriented data. In *ICAART (3)*, pages 972–979.

Jia, T., Zhao, X., Wang, Z., Gong, D., and Ding, G. (2016). Model Transformation and Data Migration from Relational Database to MongoDB. In *2016 IEEE International Congress on Big Data (BigData Congress)*, pages 60–67.

Karnitis, G. and Arnicans, G. (2015). Migration of Relational Database to Document-Oriented Database: Structure Denormalization and Data Transformation. In *2015 7th International Conference on Computational Intelligence, Communication Systems and Networks*, pages 113–118.

Kuszera, E. M., Peres, L. M., and Didonet Del Fabro, M. (2022). Exploring data structure alternatives in the rdb to nosql document store conversion process. *Information Systems*, 105:101941.

Lee, C. and Zheng, Y. (2015). SQL-to-NoSQL Schema Denormalization and Migration: A Study on Content Management Systems. In *2015 IEEE International Conference on Systems, Man, and Cybernetics*, pages 2022–2026.

Lóscio, B. F., OLIVEIRA, H. d., and PONTES, J. d. S. (2011). Nosql no desenvolvimento de aplicações web colaborativas. *VIII Simpósio Brasileiro de Sistemas Colaborativos*, 10(1):11.

Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X., and Gao, J. (2025). Large language models: A survey.

Mior, M. J., Salem, K., Aboulnaga, A., and Liu, R. (2016). NoSE: Schema design for NoSQL applications. In *2016 IEEE 32nd ICDE*, pages 181–192.

Sadalage, P. and Fowler, M. (2013). *NoSQL Essencial: Um Guia Conciso para o Mundo Emergente da Persistência Poliglota*. Novatec Editora.

Santos, M. Y. and Costa, C. (2016). Data Models in NoSQL Databases for Big Data Contexts. In Tan, Y. and Shi, Y., editors, *Data Mining and Big Data*, pages 475–485, Cham. Springer International Publishing.

Serrano, D., Han, D., and Stroulia, E. (2015). From Relations to Multi-dimensional Maps: Towards an SQL-to-HBase Transformation Methodology. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 81–89.

Stanescu, L., Brezovan, M., and Burdescu, D. D. (2016). Automatic mapping of MySQL databases to NoSQL MongoDB. In *2016 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 837–840.

Stonebraker, M., Madden, S., Abadi, D. J., Harizopoulos, S., Hachem, N., and Helland, P. (2007). The end of an architectural era (it's time for a complete rewrite). In *Proceedings of the 33rd VLDB, University of Vienna, Austria, 2007*, pages 1150–1160.

Vajk, T., Fehér, P., Fekete, K., and Charaf, H. (2013). Denormalizing data into schema-free databases. In *2013 IEEE 4th International Conference on Cognitive Infocommunications (CogInfoCom)*, pages 747–752.

Xiang Li, Zhiyi Ma, and Hongjie Chen (2014). QODM: A query-oriented data modeling approach for NoSQL databases. In *2014 IEEE WARTIA*, pages 338–345.

Zhao, G., Li, L., Li, Z., and Lin, Q. (2014). Multiple Nested Schema of HBase for Migration from SQL. In *2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pages 338–343.

Zhao, G., Lin, Q., Li, L., and Li, Z. (2014). Schema Conversion Model of SQL Database to NoSQL. In *Proceedings of the 2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, 3PGCIC '14, page 355–362, USA. IEEE Computer Society.