Avaliando MfCodeGenerator: um gerador de código para acesso a dados em bancos NoSQL

Evaluating MfCodeGenerator: a code generator for accessing data in NoSQL databases

Luan Felipe Marmentini[1], Evandro Miguel Kuszera[2]

RESUMO

Os bancos de dados relacionais são utilizados para armazenar os dados de aplicações diversas, em que os dados são organizados em tabelas compostas por linhas e colunas. No entanto, os bancos de dados relacionais não atendem de forma satisfatória cenários que demandam armazenamento de grandes quantidades de dados, com muitos acessos concorrentes e alta disponibilidade. Como alternativa surgiram os bancos de dados NoSQL, que diferem em termos de arquitetura, modelo de dados e linguagem de consulta. Esta pesquisa avaliou a ferramenta MfCodeGenerator, que faz geração de código fonte para auxiliar a migração de dados entre bases relacionais e bases NoSQL usando a linguagem Java. Foram realizados experimentos para validar o processo de geração de código e sua efetividade ao migrar os dados. Os resultados mostraram que o código gerado atende ao modelo de dados destino e pode ser usado para executar a migração dos dados.
PALAVRAS-CHAVE: Banco relacional; Geração de código; NoSQL.

ABSTRACT

Relational databases are used to store data from various applications, in which the data is organized into tables made up of rows and columns. However, relational databases do not satisfactorily meet scenarios that require storing large amounts of data, with many concurrent accesses and high availability. As an alternative, NoSQL databases emerged, which differ in terms of architecture, data model and query language. This research evaluated the MfCodeGenerator tool, which generates source code to assist in migrating data between relational databases and NoSQL databases using the Java language. Experiments were carried out to validate the code generation process and its effectiveness when migrating data. The results showed that the generated code meets the target data model and can be used to perform data migration.
KEYWORDS: Relational database; Code generation; NoSQL.

INTRODUÇÃO

Grande maioria das aplicações que manipulam dados usam bancos de dados relacionais (RDB) na camada de persistência. As propriedades ACID (atomicidade, consistência, isolamento e durabilidade) e a linguagem SQL (*Structured Query Language*) são características determinantes para escolha do modelo relacional. No entanto, para certos cenários que envolvem o gerenciamento de dados em larga escala, distribuídos e sem estrutura definida, os RDBs não são a melhor opção (STONEBRAKER et al., 2007).

[1] Aluno de Iniciação Científica. Universidade Tecnológica Federal do Paraná, Dois Vizinhos, Paraná, Brasil. E-mail: luanmarmentini@alunos.utfpr.edu.br. ID Lattes: http://lattes.cnpq.br/7068292247733084.
[2] Docente no Curso de Bacharelado em Engenharia de Software. Universidade Tecnológica Federal do Paraná, Dois Vizinhos, Paraná, Brasil. E-mail: evandrokuszera@utfpr.edu.br. ID Lattes: http://lattes.cnpq.br/5330421411544786.

Com base nisso surgiram os bancos de dados NoSQL (do inglês, *Not only SQL*) (SADALAGE; FOWLER, 2013), oferecendo uma alternativa aos bancos relacionais, diferindo em termos de arquitetura, modelo de dados e linguagem de consulta. Não há linguagem de consulta padrão, como no caso do SQL para RDB, sendo que cada banco NoSQL fornece linguagem própria.

Também não é necessário definir esquema de dados a *priori* e as entidades persistidas no banco podem variar em termos de número e tipo de atributos. Essas características trazem flexibilidade para o desenvolvimento de aplicações, mas também trazem desafios quando é necessário evoluí-la. Não é incomum o uso de diferentes bancos de dados em aplicações modernas, sendo para atender requisitos de consistência dos dados, disponibilidade, desempenho, quanto para atender requisitos relacionados à aplicação, como flexibilidade e manutenibilidade. Dois dos modelos de dados amplamente usados em aplicações modernas são o modelo relacional e o modelo orientados a documentos. Converter e migrar dados entre diferentes bancos de dados acaba se tornando uma tarefa importante, de forma que o dado seja armazenado no local mais adequado.

Há diferentes abordagens para converter e migrar dados de RDB para NoSQL. Dentre elas, o objetivo desta pesquisa visou avaliar o framework Metamorfose (KUSZERA; PERES; DIDONET DEL FABRO, 2022) no processo de migração e geração de código Java para acesso aos dados migrados, facilitando o processo de desenvolvimento de aplicações. O Metamorfose usa um esquema de dados para representar a estrutura das entidades que serão migradas do RDB para NoSQL. Esse esquema de dados pode ser usado para gerar código fonte, permitindo acessar os dados posterior a migração. O código gerado faz uso de frameworks de acesso a dados baseados em Object-NoSQL Mappers (ONMs), que permitem mapear os dados para um modelo orientado a objetos. Atualmente o Metamorfose tem suporte ao ONMs Spring Data, Impetus Kundera e Data Nucleus.

Nesta pesquisa foi realizada uma avaliação do MfCodeGenerator, módulo que recebe um esquema e gera código fonte Java. O código gerado foi validado por meio da execução de um processo de migração de dados de uma base RDB para uma base NoSQL (MongoDB). Os resultados mostram que o código gerado permitiu migrar os dados e prover acesso posterior a migração.

FUNDAMENTOS

O primeiro banco NoSQL surgiu em 1998 por Carlo Strozzi, que promovia uma distinção completa do modelo relacional (SADALAGE; FOWLER, 2013). Geralmente os bancos NoSQL são classificados em: orientados a documentos, orientados a grafos, orientados a colunas e orientados a chave-valor. Esta pesquisa foca em bancos orientados a documentos, em que os dados são organizados em documentos estruturados em JSON. Como não necessita de um esquema fixo previamente definido, proporcionam alta flexibilidade para persistir dados. O banco de dados NoSQL utilizado nos experimentos é o MongoDB, um dos bancos NoSQL mais utilizados pela comunidade [1].

Após a definição do banco de dados, uma das tarefas é desenvolver artefatos de software para acessar os dados. Há duas alternativas: uso de APIs nativas ou uso de *middlewares* para acesso aos dados, também chamados de *Object NoSQL Mappers* (ONM). Na próxima seção serão apresentados *middlewares* voltados para a linguagem Java.

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

OBJECT NOSQL MAPPERS (ONM)

As ferramentas de ONM mapeiam os documentos do banco NoSQL para linguagens orientadas a objetos. Para a linguagem Java há vários *frameworks* para a persistência de dados, como Spring Data, Data Nucleus e Impetus Kundera. O Spring tem suporte a criação de APIs REST e a persistência de dados, proporcionando desenvolver aplicações rapidamente, graças à abstração de tarefas repetitivas e complexas do desenvolvedor. O Data Nucleus[2] é um *framework* focado na manipulação de dados que cria uma camada de abstração entre a aplicação Java e banco de dados. Fornece suporte as APIs JPA, JDO e REST tornando uma opção versátil para aplicações *backend*. Impetus Kundera possibilita o mapeamento de objetos de diferentes bancos de dados para diferentes linguagens de programação. Suporta JPA e compatibilidade com bancos relacionais e NoSQL.

Mesmo com a existência destes *frameworks*, o trabalho do desenvolvedor ainda pode ser tedioso e propenso a erros. É interessante uma ferramenta que faça a geração de código de acesso a dados, sugerindo quais anotações e configurações são necessárias para criar código funcional e livre de erros.

MFCODEGENERATOR

MfCodeGenerator[3] gera as classes que mapeiam os documentos, utilizando como referência um esquema NoSQL que denota a estrutura das entidades e seus relacionamentos. A Figura 1 apresenta o fluxo de execução do MfCodeGenerator. A partir de um esquema NoSQL e do tipo de ONM alvo, a ferramenta gera um conjunto de classes Java para representar as entidades NoSQL. As entidades são anotadas conforme o ONM selecionado, sendo que as anotações são escolhidas pelo MfCodeGenerator e podem ser customizadas pelo desenvolvedor, com novos elementos.

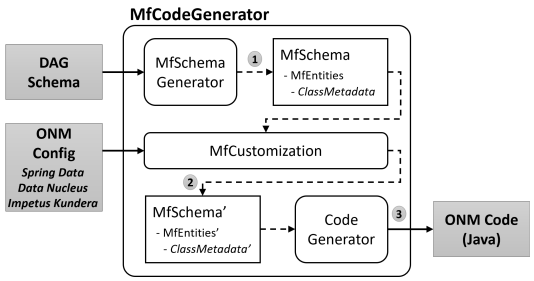

Figura 1 – Arquitetura do MfCodeGenerator

O código gerado deve ser adicionado em um projeto Java para permitir acesso aos dados no banco NoSQL. Além disso, a ferramenta permite adicionar customizações ao código fonte gerado, sendo possível estender para novos ONMs e linguagens de programação.

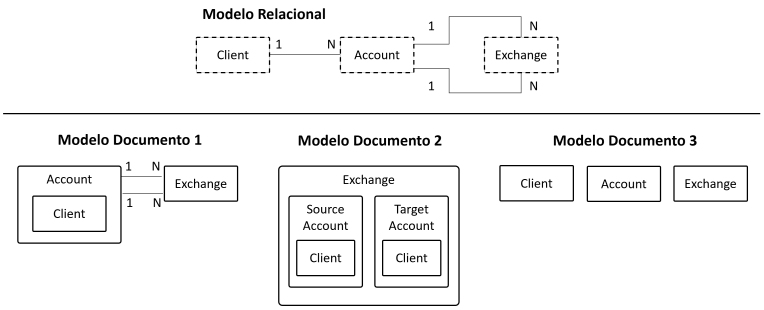

Figura 2 – Modelo de dados relacional e modelos de dados orientados a documentos usados no experimento.

EXPERIMENTOS

Para avaliar o processo de geração de código da ferramenta MfCodeGenerator e posterior migração de dados entre bancos relacionais (RDBs) e não relacionais (NoSQLs), foram executados testes com diferentes formas de organização dos dados, a fim de demonstrar e comparar as diferentes abordagens possíveis. Neste experimento foi gerado código para o Spring Data. Na parte superior da Figura 2 é apresentado o diagrama entidade-relacionamento do banco de dados relacional utilizado nos experimentos. Ele tem três tabelas: *Client*, *Account* e *Exchange*. Cada tabela tem 1000 registros que foram gerados aleatoriamente. Na parte inferior da Figura 2 há três modelos orientados a documentos. Cada retângulo representa uma coleção de documentos e retângulos internos representam documentos aninhados. O modelo Documento 3 é semelhante ao relacional, sem aninhamento entre entidades. Com base nos modelos de dados foram realizados experimentos envolvendo as seguintes etapas: **1) Planejamento do modelo de dados NoSQL**: O MfCodeGenerator define um esquema NoSQL por meio de DAGs (*Directed Acyclic Graphs*), que denotam a representação da organização dos dados no modelo orientado a objetos, é através dele que a ferramenta será capaz de gerar o código fonte desejado. Nesta etapa, foi utilizado o QBMetrics[4] para criar os DAGs com base nos esquemas da Figura 2. **2) Geração do código**: O MfCodeGenerator recebe um DAG de entrada e gera classes Java de acordo com o ONM escolhido pelo usuário. É importante notar que o MfCodeGenerator sugere as anotações necessárias para as entidades e seus relacionamentos. **3) Migração dos dados**: Neste etapa o código gerado é usado para migrar os dados do RDB para o MongoDB. Os dados são carregados do RDB para o modelo orientado a objetos e, depois, são persistidos no MongoDB como documentos JSON. **4) Análise dos resultados**: nesta etapa são avaliados o código gerado e também o resultado da migração dos dados.

RESULTADOS

Abaixo são apresentados os resultados obtidos depois de gerar código usando MfCodeGenerator para os esquemas NoSQL apresentados na Figura 2 e execução da migração de dados.

Para o **Esquema Documento 1** os dados foram dispostos em duas coleções, uma para

[2] https://www.datanucleus.org/
[3] https://github.com/evandrokuszera/metamorfose-code-generator
[4] https://github.com/evandrokuszera/nosql-query-based-metrics

```java
@Document(collection = "Account")
public class Account{
  private Integer id;
  private Integer id_client;
  private Double value;
  private Client client;
  @Id
  private String _id;
 }

public class Client{
  private Integer id;
  private String cpf;
  private String name;
  private String address;
  private String phone;
 }
```

```java
@Document(collection = "Exchange")
public class Exchange{
  private Integer id;
  private Integer id_conta_source;
  private Integer id_conta_dest;
  private Double value;
  @Id
  private String _id;
  @DocumentReference(lookup="{'id':?#{#self.
     id_conta_source}}")
  private Account accountSource;
  @DocumentReference(looku ="{'id' ?#{#self.
     id_conta_dest}}")
  private Account accountDest;
 }
```

Figura 3 – Códigos gerados no primeiro experimento

```java
@Document(collection = "Exchange")
public class Exchange{
  private Integer id;
  private Integer id_conta_source;
  private Integer id_conta_dest;
  private Double value;
  private Account accountSource;
  private Account accountDest;
  @Id
  private String _id;
}
```

```java
public class Client{
  private Integer id;
  private String cpf;
  private String name;
  private String address;
  private String phone;
}

public class Account{
  private Integer id;
  private Integer id_client;
  private Double value;
  private Client client;
}
```

Figura 4 – Códigos gerados no segundo experimento

Account, possuindo um objeto *Client* embutido. A outra coleção armazena os dados das transações, esses objetos além das informações possuem referências as contas origem e destino. A Figura 3 mostra o código gerado pelo MfCodeGenerator. Para o **Esquema Documento 2**, os dados foram dispostos em uma única coleção, embutido as demais entidades. (*Exchange*) foi escolhida como entidade raiz. As contas de destino e origem são embutidas em *Exchange*. As contas, por sua vez, possuem o objeto cliente (*Client*) embutido. O código gerado está apresentado na Figura 4. Para o **Esquema Documento 3** três coleções foram criadas, uma para contas, possuindo uma referência a um cliente, uma para clientes, com um vetor de referências às contas e finalmente uma coleção de objetos transação que possui duas referências, uma para a conta destino e outra para a conta origem foram criadas. O código gerado é apresentado na Figura 5.

Após o processo de geração de código foram executados testes para migração de dados do RDB para MongoDB. Na sequência, foi verificada a consistência dos dados migrados. Com base na análise realizada foi possível verificar que todos os dados foram migrados, mantendo a consistência de dados.

CONSIDERAÇÕES FINAIS

Após a realização dos experimentos, foi possível observar que o processo de migração de dados é uma tarefa trabalhosa e possui muitos riscos de integridade dos dados, por isso o uso de ferramentas auxiliares é crucial. A ferramenta MfCodeGenerator foi capaz de gerar códigos Java compatíveis com o Spring Data, automatizando um processo que seria realizado manualmente. Com

```
@Document(collection = "Client")
public class Client{
   private Integer id;
   private String cpf;
   private String name;
   private String address;
   private String phone;
   @Id
   private String _id;
}

@Document(collection = "Exchange")
public class Exchange{
   private Integer id;
   private Integer id_conta_source;
   private Integer id_conta_dest;
   private Double value;
   @Id
   private String _id;
}
```

```
@Document(collection = "Account")
public class Account{
   private Integer id;
   private Integer id_client;
   private Double value;
   @Id
   private String _id;
   @DocumentReference(lookup="{'id':?#{#self.id_client}}"
      )
   private Client client;
   @ReadOnlyProperty
   @DocumentReference(lookup="{'id_conta_dest':?#{#self.
      id} }")
   private List<Exchange> exchange;
}
```

Figura 5 – Códigos gerados no terceiro experimento

tudo, seu funcionamento depende da produção do DAG, que por não ser intuitiva e demasiadamente complexa, dificulta o inicio da atividade de migração de dados. Sabendo disso, o suporte à integrações com ferramentas semelhantes que forneçam interfaces gráficas facilitaria sua utilização, além de padronizar as configurações, auxiliando na curva de aprendizado do desenvolvedor. Como trabalho futuro pretende-se realizar experimentos para avaliar os demais ONMs e suporte a modelos com diferentes tipos de aninhamentos e relacionamentos.

AGRADECIMENTOS

Este trabalho foi realizado com o apoio da Universidade Tecnológica Federal do Paraná.

CONFLITO DE INTERESSE

“Não há conflito de interesse”.

REFERÊNCIAS

KUSZERA, Evandro Miguel; PERES, Leticia Mara; DIDONET DEL FABRO, Marcos. Exploring data structure alternatives in the RDB to NoSQL document store conversion process. **Information Systems**, v. 105, p. 101941, 2022. ISSN 0306-4379.

SADALAGE, P.J.; FOWLER, M. **NoSQL Essencial: Um Guia Conciso para o Mundo Emergente da Persistência Poliglota**. [S.l.]: Novatec Editora, 2013. ISBN 9788575223383. Disponível em: .

STONEBRAKER, Michael et al. The End of an Architectural Era (It’s Time for a Complete Rewrite). In: PROCEEDINGS of the 33rd VLDB, University of Vienna, Austria, 2007. [S.l.: s.n.], 2007. P. 1150–1160.