Uma Aplicação para Migração de Dados de Banco Relacional para MongoDB

Rodrigo Machado[1], **Gustavo Girardon**[1], **Victor Costa**[1], **Maicon Bernardino**[1], **Marcus Jacomé**[1], **Robson Gonçalves**[1]

[1] Universidade Federal do Pampa (UNIPAMPA)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{ggirardon, victorsc.rs, rodrigo.blizzard92}@gmail.com
bernardino@acm.org, {robson.o.goncalves, marcus_norberto}@gmail.com

Abstract. *In order to remain competitive in the market, it is important that organizations that develop software are constantly adapting to new technologies. One technology that has caught the attention of developers is NoSQL databases. Many organizations collect large volumes of data, and often store them in relational databases. This paper presents a solution designed to migrate data from a relational database to a nonrelational one.*

Resumo. *Para manter a competitividade no mercado, é importante que as organizações que desenvolvem softwares estejam em constante adaptação no que se refere a novas tecnologias. Uma tecnologia que vem chamando a atenção dos desenvolvedores são os bancos de dados NoSQL. Muitas organizações coletam grandes volumes de dados, e frequentemente armazenam em bancos de dados relacionais. Esse artigo apresenta uma solução desenvolvida com a finalidade de migrar os dados de um banco de dados relacional para um não-relacional.*

1. Introdução

Com a rápida evolução de tecnologias para o desenvolvimento de software, é crucial que as organizações estejam em constante adaptação para manterem a competitividade no mercado. Muitas dessas organizações coletam grandes volumes de dados, que frequentemente são armazenados em bancos de dados relacionais.

Os banco relacionais são amplamente utilizados, porém eles não são capazes de armazenar e processar dados grandes de forma eficaz e não são muito eficientes para fazer transações e participar de operações [Abramova and Bernardino 2013]. Para superar alguns desses problemas, surgiu um novo paradigma, os bancos de dados *NoSQL*, que são mais adequados para o uso em ambientes web.

Os dados acumulados pelos sistemas de informação são um dos ativos importantes para a maioria das empresas. Pressionadas pelas demandas dos clientes e pelas mudanças tecnológicas, as empresas, de tempos em tempos, migram de um sistema de informação para outro. Portanto, os dados do sistema legado precisam ser migrados para o novo sistema [Karnitis and Arnicans 2015].

Este artigo apresenta uma aplicação de migração de dados de um banco relacional para um banco de dados *NoSQL*, desenvolvida dentro de um projeto no setor de desenvolvimento de uma organização.

2. Fundamentação Teórica

Os bancos de dados relacionais foram introduzidos na década de 1970 para permitir que os aplicativos armazenassem dados por meio de uma linguagem de consulta e modelagem de dados padrão [Codd 1970].

Esses bancos de dados armazenam dados como um conjunto de tabelas, cada uma com informações diferentes. Todos os dados estão relacionados para que seja possível acessar informações de diferentes tabelas simultaneamente.

Na organização onde este trabalho foi desenvolvido, o principal Sistema de Gerenciamento de Banco de Dados utilizado é o IBM DB2, que é relacional e fornece recursos avançados de gerenciamento de dados e de análise para suas cargas de trabalho transacionais e de *warehousing*.

Como os sistemas tem cada vez mais usuários e dados a serem salvos, alguns problemas dos bancos relacionais tornaram-se evidentes. Um deles é o desempenho do banco de dados relacionado ao acesso a dados e à estrutura básica do modelo relacional. O SQL permite uma fácil extração de dados, mas quando o volume de informações é enorme, o tempo de execução da consulta pode ficar lento [Jayathilake et al. 2012].

Para mitigar este e outros problemas, muitas organizações estão utilizando bancos de dados *NoSQL*. O *NoSQL* engloba uma ampla variedade de diferentes tecnologias de banco de dados, mas geralmente todos os bancos de dados *NoSQL* têm alguns recursos em comum. Como por exemplo: esquemas dinâmicos; replicação; e cache integrado [Leavitt 2010].

A organização onde o desenvolvimento da aplicação foi realizado, definiu a utilização do banco de dados MongoDB. O MongoDB permite que um objeto JSON seja armazenado, o qual pode ser facilmente convertido em objetos JavaScript no código. Essa simplicidade e flexibilidade tornam o uso do MongoDB mais fácil para muitos programadores, sem a necessidade de entender o SQL [Lawrence 2014].

3. Aplicação de Migração de dados DB2 para MongoDB

A aplicação *DB2toMongo* foi desenvolvida dentro do contexto de um projeto de desenvolvimento de um sistema genérico para consulta a dados utilizando API.

O projeto foi desenvolvido por cinco estagiários, que ficaram responsáveis por diferentes partes da arquitetura do projeto, como mostra a Figura 1.

3.1. Requisitos da Aplicação

Os requisitos foram definidos pelo *Product Owner*, e passados como uma lista de funcionalidades que a aplicação deveria atender. Os requisitos definidos foram:

1. A aplicação deve ler os dados do DB2 conforme *query* previamente definida pelo usuário;
2. A aplicação deve migrar os dados lidos pela *query* para o banco de dados *MongoDB*;
3. A aplicação deve trabalhar em dois modos escolhidos pelo usuário:
    (a) **Automático**: A aplicação deve executar a *query* realizando a comparação de data por meio de uma coluna pré-estabelecida pelo usuário. O usuário

também deve definir uma coluna para ser chave primária da sua *query*. A partir disso, a aplicação deve comparar as datas, lendo no DB2 tudo que foi inserido ou modificado em até um dia antes da última execução e inserir os dados inexistentes ou alterar os dados que já tem a chave primária definida no *MongoDB*;

(b) **Substituição**: A aplicação deve apagar todos os dados existentes no *MongoDB* e realizar novamente a migração dos dados do DB2.

4. A aplicação deve salvar a data de sua última execução e o usuário deve poder alterar essa data caso necessite;
5. Deve ser possível adicionar quantas *querys* o usuário desejar em uma única execução;
6. As configurações não podem ser definidas dentro do código-fonte.

3.2. Desenvolvimento e Testes

Como processo de desenvolvimento e gerenciamento de equipe foi adotado o processo ágil *Scrum*, devido ás suas inúmeras vantagens e à afinidade dos membros da equipe com a sua abordagem. Assim como processos tradicionais, o *Scrum* também possui papéis, artefatos e atividades bem definidas.

A aplicação foi desenvolvida respeitando todos os requisitos citados na Subseção 3.1 e diversos testes manuais foram realizados, com diferentes *querys*, a fim de verificar todas as funcionalidades.

Para facilitar a configuração de dependências, foi utilizado o *Maven*, que é uma ferramenta amplamente utilizada para gerenciamento e compreensão de projetos de software.

3.3. Arquitetura

A Figura 1, apresenta uma visão geral da arquitetura da solução desenvolvida dentro da organização, destacando a aplicação apresentada por este trabalho.

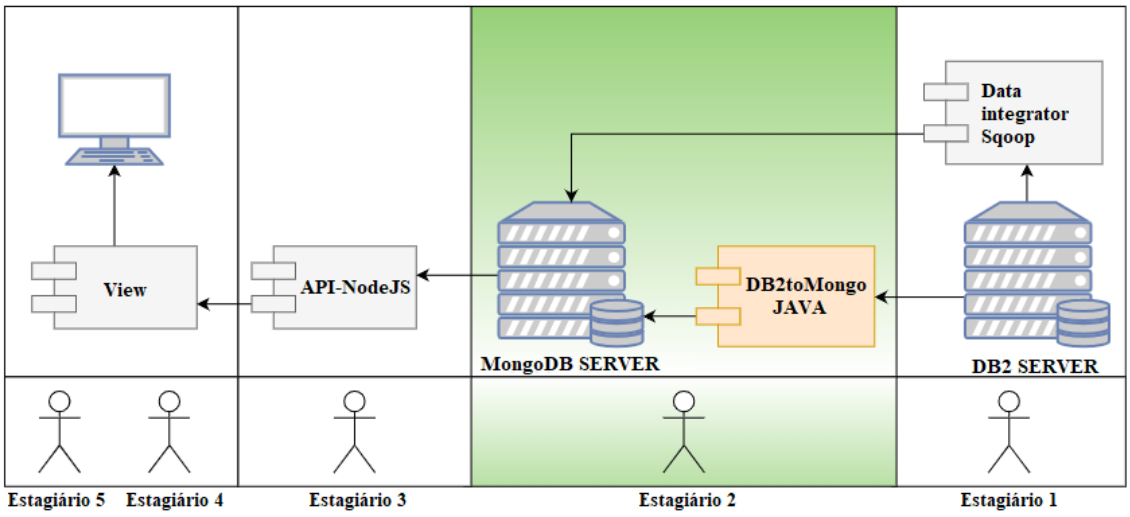

Figura 1. Arquitetura da Solução Completa

4. Exemplo de Uso

Na atual versão da aplicação DB2Mongo, ela é capaz de ler um banco DB2 (relacional) e exportar suas tabelas para o banco MongoDB (não-relacional) como coleções. A aplicação também faz a comparação dos dados já existentes no banco, para realizar a atualização dos mesmos. A aplicação trabalha com dois modos (escolhidos através do arquivo de configuração):

1. **AUTO**: Neste modo, a aplicação utiliza a data de alteração de determinado registro de uma tabela para comparar se deve ou não realizar uma alteração ou inserção. Ela analisa todos os registros inseridos ou alterados até um dia antes do momento de sua execução.
2. **REPLACE**: Neste modo, todos os documentos do *MongoDB* são deletados, e são inseridos novamente todos os registros selecionados por determinada *query*.

4.1. Configurações

As configurações devem ser realizadas através do arquivo ***config.properties*** (Figura 2). O arquivo contém as seguintes configurações:

- **MONGODB CONNECTION CONFIG**: Nesta seção do documento devem ser preenchidas as seguintes informações de conexão com o *MongoDB*:
    1. Usuário;
    2. Senha;
    3. Host;
    4. Porta.

- **RELATIONAL DATABASE CONNECTION CONFIG**: Nesta seção do documento devem ser preenchidas as seguintes informações de conexão com o DB2:
    1. Nome da classe JDBC (já configurada para db2);
    2. *URL* de conexão (exemplo: jdbc:db2://host:porta/db);
    3. Usuário;
    4. Senha.

- **DATABASE MONGO CONFIG**: Nesta seção do documento é definida a *database* que será utilizada dentro do *MongoDB*.

- **QUERY CONFIG**: Nesta seção do documento, a primeira coisa a ser definida é o número de *querys* que devem ser utilizadas na execução da aplicação. Logo após, para cada *query* devem ser preenchidos os seguintes campos:
    1. *Query* (Exemplo: Select * from table);
    2. Coleção que será criada no mongo para corresponder a *query*;
    3. A coluna correspondente a data de alteração (para verificação no *update*);
    4. A coluna correspondente a chave primária da coleção.

    Todos os campos descritos acompanham um número dentro da documentação, que corresponde a ordem de execução da *query*. O campo *Query* está descrito no documento de configurações como: prop.query.**x** (o **x** corresponde ao número associado a *query*).

Logo, se o usuário quer que determinada *query* seja executada primeiro, deverá colocar todos os campos relacionados a *query* de maneira: prop.query.**1**; prop.mongo.collection.**1**; prop.key.of.date.**1**; prop.pkey.**1**. Vale ressaltar que podem ser adicionadas quantas *querys* se desejar, e todas devem ter uma numeração para serem reconhecidas pela aplicação.

- **SELECT THE TYPE OF OPERATION: (AUTO : 1) (REPLACE: 2)** : Por último, deve ser definido o tipo de operação que será executada pelo sistema. A chave 1 refere-se a opção **AUTO** e a chave 2 a opção **REPLACE**.

```
# MONGODB CONNECTION CONFIG
prop.server.mongo.user = admin
prop.server.mongo.pwd = tibia123
prop.server.mongo.host = 10.12.0.47
prop.server.mongo.port = 27017

# RELATIONAL DATABASE CONNECTION CONFIG

prop.db.jdbcClassName = com.ibm.db2.jcc.DB2Driver
prop.db.url = jdbc:db2://10.12.0.57:50000/dbgurit
prop.db.user = rodrigo
prop.db.password = hIt6axQF

# DATABASE  MONGO CONFIG
prop.mongo.database = admin

# QUERY CONFIG
prop.number.of.querys = 3

prop.query.1 = SELECT * FROM ERP_LOG.SYSLOG
prop.mongo.collection.1 = admin
prop.key.of.date.1 = DT_HR_INCLUSAO
prop.pkey.1 = ID_SYSLOG


prop.query.2 = SELECT * FROM ERP.INSCRICAO_PROCSEL
prop.mongo.collection.2 = NovaCollection
prop.key.of.date.2 = DT_ALTERACAO
prop.pkey.2 = ID_INSCRICAO_PROCSEL
```

Figura 2. Exemplo do Documento de Configuração da Aplicação DB2toMongo

4.2. Classes e Métodos

A aplicação java desenvolvida conta com 6 classes com diferentes responsabilidades, conforme apresenta a Figura 3.

- **Config.java**: Contém apenas um método que é responsável por retornar o arquivo ***config.properties***.

- **DB2Connection.java**: Conta com o método ***connectionDB2()*** que é responsável por realizar a conexão com o banco de dados DB2, conforme configurações definidas.

- **DB2Read.java**: Essa classe é responsável pela realização da leitura no Banco de Dados DB2, e contém 4 métodos:
    1. ***readDB()*** : método que retorna a uma lista contendo todos as ocorrências de uma determinada *query*. Cada posição da lista contém a ocorrência em uma determinada coluna.
    2. ***readDBColumns()***: método que retorna a uma lista contendo os nomes das colunas de uma determinada *query*.
    3. ***readDBforUpdate()*** : método que retorna a uma lista contendo todos as ocorrências de uma determinada *query*, utilizando a data de execução como referência (***executeDate.txt***). Cada posição da lista contém a ocorrência em uma determinada coluna.
    4. ***updateDate()*** : método que responsável por alterar a data final da execução existente no arquivo ***executeDate.txt***. Na atual versão dessa aplicação, foi padronizado que seriam lidos do banco relacional arquivos que tivessem sido alterados até um dia antes da execução da aplicação.

- **MongoConnection.java**: Conta com o método ***connectionMongo()*** que é responsável por realizar a conexão com o banco de dados *MongoDB*, conforme configurações definidas.

- **MongoInsert.java**: Classe responsável por realizar as inserções e atualizações no Banco de Dados *MongoDB*. A classe contém os seguintes métodos:
    1. ***insertMongo()***: Método responsável por inserir no Banco de Dados *MongoDB* todos os registros retornados por uma determinada *query*, sem qualquer tipo de verificação.
    2. ***replaceMongo()***: Método responsável por deletar todos os registros de uma coleção, e substituir por todos os registros retornados por uma determinada *query*. Com isso, todo o banco é substituído.
    3. ***updateMongo()***: Método responsável por inserir ou atualizar determinado documento no Banco de Dados MongoDB. Esse método verifica se já existe o registro baseado em sua chave primária (definida nas configurações da *query*, e caso não exista ele insere um novo documento. Caso a chave já exista, ele atualiza o documento no *MongoDB* conforme atualizações retornadas pelo DB2.

- **Execute.java**: Classe responsável pela execução da aplicação. Além do método ***main()***, que utiliza o tipo de operação definida no ***config.properties*** para definir a sua execução, também contém mais dois métodos:
    1. ***getDateTime()***: Método responsável por capturar a data e hora de execução da aplicação.
    2. ***saveDate()***: Método responsável por salvar no arquivo **executeDate.txt** a data capturada.

5. Considerações Finais

Os sistemas estão ficando cada vez mais complexos e com mais volume de dados, o que torna os sistemas de banco de dados relacionais menos vantajosos para o desenvolvimento

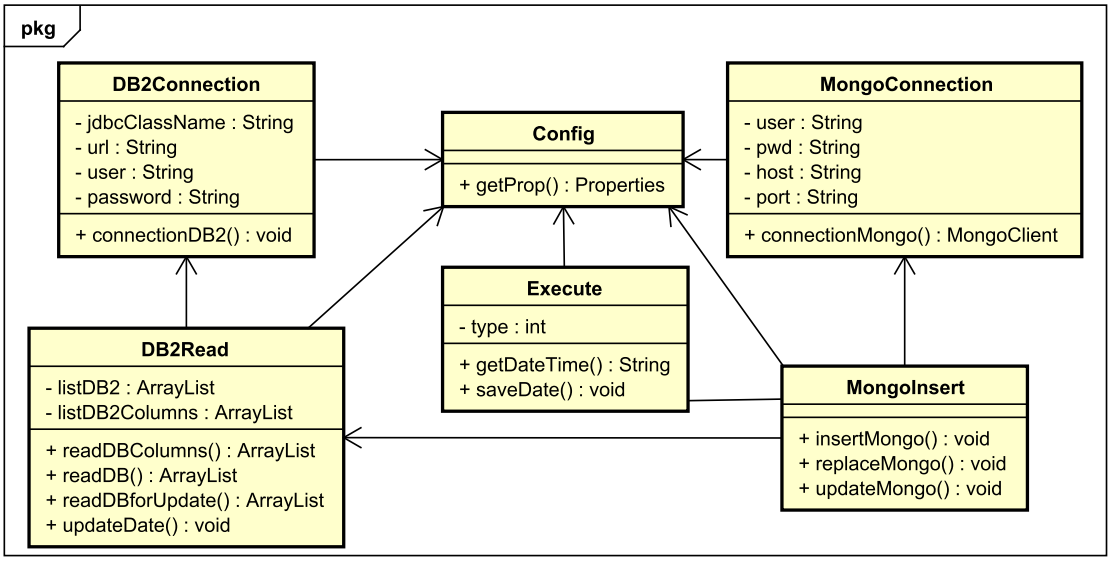

Figura 3. Diagrama de Classe da Aplicação DB2toMongo

de novos sistemas. Em contrapartida, sem a automatização de migração de um banco de dados legado, as organizações acabam escolhendo permanecer com o sistema de gerenciamento de banco de dados que já vem sendo utilizado.

Este trabalho apresentou uma possível solução para que os desenvolvedores de software consigam realizar a migração de dados automatizada de um banco de dados relacional para o MongoDB.

Por fim, foi possível utilizar essa aplicação específica no contexto do projeto executado pela organização, preenchendo uma lacuna existente dentro da organização para a realização da migração de dados do IBM DB2 para uma tecnologia que a organização citada tem interesse em adicionar em projetos futuros.

Referências

Abramova, V. and Bernardino, J. (2013). Nosql databases: Mongodb vs cassandra. In *Proceedings of the International C* Conference on Computer Science and Software Engineering*, C3S2E '13, pages 14–22, New York, NY, USA. ACM.

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387.

Jayathilake, D., Sooriaarachchi, C., Gunawardena, T., Kulasuriya, B., and Dayaratne, T. (2012). A study into the capabilities of nosql databases in handling a highly heterogeneous tree. In *2012 IEEE 6th International Conference on Information and Automation for Sustainability*, pages 106–111. IEEE.

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. IEEE.

Lawrence, R. (2014). Integration and virtualization of relational sql and nosql systems including mysql and mongodb. In *2014 International Conference on Computational Science and Computational Intelligence*, volume 1, pages 285–290. IEEE.

Leavitt, N. (2010). Will nosql databases live up to their promise? *Computer*, 43(2):12–14.