

Gerenciamento de mudanças em banco de dados: Um estudo de caso utilizando Flyway e Liquibase

Ismael Carlos de Almeida Figueiredo, Daves Martins

Curso Bacharelado em Sistemas de Informação – Centro de Ensino Superior de Juiz de Fora (CES/JF) Caixa Postal 36.016-000 – Juiz de Fora – MG – Brasil

{ismaelcaf,davesmartins}@gmail.com

***Abstract:** During the life of a software, the database or databases undergo constant changes, as well as the source code. But while the source code has robust tools, such as GIT and SVN to manage your changes, managing database changes are often not tool-assisted, requiring manual labor. The purpose of this article is to present the Flyway and Liquibase tools as a solution to automate the application of changes to databases. A case study will be described aiming to show the characteristics and differences of the tools in order to serve as the basis for a decision of use among them.*

***Resumo:** Durante a vida de um software, a base ou as bases de dados sofrem mudanças constantes, assim como o código-fonte. Mas enquanto o código fonte possui ferramentas robustas, como o GIT e o SVN para gerenciar suas mudanças, gerenciar as mudanças do banco de dados costumam não serem assistidas por ferramentas, sendo necessário trabalho manual. O objetivo deste artigo apresentar as ferramentas Flyway e Liquibase como solução para automatizar a aplicação das mudanças nas bases de dados. Será descrito um estudo de caso com o objetivo mostrar as características e diferenças das ferramentas com intuito a servir de base para um decisão de uso entre elas.*

1. INTRODUÇÃO

Segundo SOMMERVILLE (2011) a mudança é uma realidade para grandes sistemas. As necessidades e requisitos organizacionais se alteram durante a vida útil de um sistema, bugs precisam ser reparados e os sistemas necessitam se adaptar às mudanças em seu ambiente. Para garantir que as mudanças sejam aplicadas ao sistema

de uma forma controlada, você precisa de um conjunto de processos de gerenciamento de mudanças, apoiado por ferramentas.

Segundo ZIMERMANN, SOUSA e SPÍNOLA (2018) na constante busca por melhoria na qualidade dos softwares, muitas empresas adotam um processo de gerenciamento de mudanças devido a sua importância para mensurar o impacto de tais mudanças. Para Pressman (apud ZIMERMANN; SOUSA; SPÍNOLA, 2018) a falta de gerenciamento das mudanças gera dificuldade de se rastrear a evolução do software.

Embora o processo de gerenciamento das mudanças do código fonte seja mais comumente usado, por exemplo temos ferramentas como o GIT e o SVN que ajudam nessa tarefa. Para o banco de dados as mudanças geralmente são feitas em um processo à parte. Segundo JOTZ (2018), o modelo tradicional de evolução do banco de dados requer que os arquivos com as instruções sejam executados de forma manual em cada um dos ambientes, por um DBA (*Database administrator*), que é o profissional responsável por gerenciar bancos de dados. Às vezes para se executar uma operação simples seja preciso esperar um longo tempo devido a disponibilidade do(s) DBA(s), podendo se tornar um gargalo.

No ambiente de desenvolvimento Java, que é o foco deste trabalho, existem muitos frameworks de software livre que podem ajudar no gerenciamento da evolução do banco de dados. Segundo (KÖBLER, 2018), os dois principais são Liquibase e Flyway. Ambas são liberadas sob a licença Apache 2 e projetadas para suportar sistemas de bancos de dados relacionais, como Oracle, DB2, SQL Server, MySQL, PostgreSQL, H2, HSQL e Apache.

Este artigo está dividido da seguinte forma: Na seção 2 serão apresentados problemas rotineiros no gerenciamento de mudanças em bancos dados, bem como, a apresentação de ferramentas para automatizar aplicação dessas mudanças no banco de dados como solução para minimizá-los. Na seção 3 serão apresentadas as ferramentas Flyway e Liquibase demonstrando sua configuração e funcionamento básico. Na seção 4 será apresentado um estudo de caso onde será feita uma pequena implementação onde as ferramentas Flyway e Liquibase foram utilizadas para fazer as mudanças no banco de dados. O intuito é mostrar as principais características e diferenças das ferramentas de

forma a ajudar em uma possível escolha de uso entre elas. Na seção 5 serão feitas as considerações finais e trabalhos futuros.

2. GERENCIAMENTO DE MUDANÇAS EM BANCO DE DADOS

Segundo DATICAL (2018) os DBAs se queixam que lutam constantemente para entender o estado do banco de dados em ambientes diferentes e muitas vezes complexos. Saber quais alterações foram executadas ou não em cada um dos ambientes requer uma quantidade significativa de análise manual. Esta falta de visibilidade, juntamente com a pressão para acompanhar o volume de mudanças que surgem do desenvolvimento, resultam em mudanças feitas fora do processo. Isso causa ambientes de banco de dados desatualizados ou inconsistentes, o que leva a mais erros na sua implantação, maior tempo gasto na solução de problemas e um aumento de retrabalho.

2.1. PROBLEMAS COMUNS

Um cenário comum é relatado por KÖBLER (2018):

Uma equipe vem desenvolvendo e modificando o código, enviando e/ou recuperando alterações para o Sistema de Controle de Versão, atualizando o ambiente de desenvolvimento e reiniciando o aplicativo. De repente, alguém se Queixa: - Quem mudou alguma coisa no banco de dados? E por que não fui informado sobre quais alterações fizeram? Agora o aplicativo está me dando uma mensagem de erro, e estou tendo que ajustar meu acesso ao banco de dados! Esse é o tipo de problema e frustração que uma equipe realmente gostaria de evitar. (p. 1, tradução nossa).

Outro cenário citado por KÖBLER (2018):

Uma equipe foi instruída a solucionar problemas na versão A, que está em uso de produção no sistema de um cliente. No entanto, o trabalho de desenvolvimento do mesmo software já chegou à versão B, e extensas alterações no banco de dados foram feitas. Infelizmente nada foi documentado adequadamente, já que a equipe estava focada em terminar o desenvolvimento e ninguém se preocupou com o momento em que fosse necessário corrigir um problema na versão antiga. Onde a equipe pode encontrar agora o layout do banco de dados que corresponde à versão A? O código está disponível no sistema de controle de versão; Afinal, os desenvolvedores não foram imprudentes. Mas o banco de dados? Por que o banco de dados não está vinculado ao sistema de controle de versão? (p. 1, tradução nossa).

2.2. OBJETIVO

Dado o cenário de problemas descrito acima, o objetivo deste artigo é caminhar para uma solução mesmo que parcial do problema de gerenciamento de mudanças de bancos de dados, onde o foco será nas ferramentas que podem ser utilizadas. Essas ferramentas podem definir regras e automatizar passos que são parte de um processo.

Segundo JOTZ (2018) em torno de 70% do tempo dos DBAs de uma grande organização é gasto para aplicar atualizações de bancos de dados providas do processo de desenvolvimento de um software. Visto isso, ferramentas que possam automatizar esse processo tornando-o mais simples, podem poupar muito tempo.

Somente o fato de usar ferramentas para automatizar a aplicação de alterações no banco de dados não engloba todo o processo de gerenciamento de mudanças de banco dados. Porém isso pode significar um passo a caminho da melhora desse gerenciamento, o que pode implicar em ganhos de tempo significativos no dia a dia das organizações.

Segundo APOLINÁRIO, QUEIRÓS e CRUZ (2015), a adoção da ferramenta Liquibase trouxe mais agilidade e flexibilidade para o ambiente de desenvolvimento de um sistema de uso interno da instituição EMBRAPA e também atendeu a necessidade de atualizações incrementais no banco de dados de produção. O que indica que as ferramentas que serão mostradas neste artigo podem de fato ajudar a diminuir o tempo gasto com gerenciamento das mudanças no banco de dados no dia a dia das organizações.

Visto isto, a apresentação de ferramentas que será descrita nesse artigo, pode ser entendida como satisfatória como um passo a caminho da solução do problema da gerência de mudanças de banco dados, devido aos benefícios de seu uso. A seguir vamos ver os Frameworks Flyway e Liquibase, bem como seu funcionamento básico.

3. FERRAMENTAS PARA GERENCIAMENTO DE MUDANÇAS DE BANCO DE DADOS

Frameworks gerenciadores de mudanças de banco de dados são ferramentas que se encarregam de gerir as mudanças que devem ser realizadas no banco de dados de

uma aplicação. Eles são responsáveis manter as alterações versionadas junto com o código fonte, bem como verificar o estado ou versão do banco dados e aplicar as alterações necessárias, observando a ordem correta de execução. FLYWAY (2018)

A seguir serão apresentados os Frameworks Flyway e Liquibase, demonstrando seu funcionamento básico e mostrando seus principais pontos fortes e fracos. Outros frameworks similares a eles com desenvolvimento em evolução e número de usuários relevante, com base em buscas na internet, não foram encontrados.

3.1. FLYWAY

O Flyway foi concebido desde o princípio com o objetivo de ser fácil de configurar e usar. Ele preza a simplicidade e convenção ao invés de configuração. FLYWAY (2018).

Para explicar o funcionamento do Flyway vamos imaginar o cenário mais simples, que é quando o flyway é apontado para um banco de dados vazio. O Flyway vai tentar localizar a sua tabela de controle da evolução do esquema, como o banco de dados está vazio, ele não vai conseguir localizá-la, então será necessário criar uma tabela chamada *flyway_schema_history*. Essa tabela será usada para gravar os registros de alteração no schema do banco de dados. Esse processo pode ser visto na figura 1.

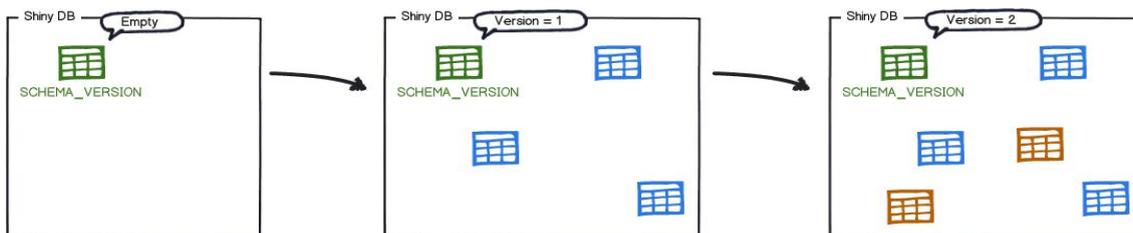


Figura 1. Aplicando as *migrations* no banco de dados - Fonte: Flyway

Após a tabela *flyway_schema_history* estar criada, o flyway vai começar a buscar no sistema de arquivos ou classpath da aplicação por *migrations*, que são alterações de banco de dados que podem ser escritas em SQL ou Java. As *migrations* são ordenadas pelo número de versão e são aplicadas seguindo essa ordem. O número de versão é determinado pelo nome do arquivo, que deve obedecer um padrão de nomenclatura estabelecida. Por exemplo, o arquivo "*V1__Create_person_table.sql*", no qual, "*V*" indica que é uma *migration* do Flyway, "*1*" é o número sequencial da

alteração, “__” é convenção do Flyway e “*Create_person_table*” é uma descrição da alteração contida no arquivo.

Como todas *migrations* foram aplicadas a tabela *flyway_schema_history* ficará como mostrado na figura 2.

flyway_schema_history

| installed_rank | version | description | type | script | checksum | installed_by | installed_on | execution_time | success |
|----------------|---------|---------------|------|-----------------------|------------|--------------|--------------------------|----------------|---------|
| 1 | 1 | Initial Setup | SQL | V1__Initial_Setup.sql | 1996767037 | axel | 2016-02-04 22:23:00.0 | 546 | true |
| 2 | 2 | First Changes | SQL | V2__First_Changes.sql | 1279644856 | axel | 2016-02-06 09:18:00.0 | 127 | true |

Figura 2. Estado da tabela *flyway_schema_history* - Fonte: Flyway

Sempre que for necessário fazer uma alteração no banco de dados, seja ela na estrutura ou dados, basta adicionar uma *migration* com número de versão maior do que a atual. Na próxima vez que o Flyway iniciar ele irá localizá-la e a aplicar no banco de dados.

Existem várias formas de configurar o Flyway em um projeto; por linha de comando, API Java, Maven ou Gradle (Gerenciadores e automatizadores de *Builds* ou construção de projeto). Um exemplo da integração através da API java pode ser visto na figura 3.

```
package foobar;

import org.flywaydb.core.Flyway;

public class App {
    public static void main(String[] args) {
        // Create the Flyway instance
        Flyway flyway = new Flyway();

        // Point it to the database
        flyway.setDataSource("jdbc:h2:file:./target/foobar", "sa", null);

        // Start the migration
        flyway.migrate();
    }
}
```

Figura 3. Integração do flyway usando a API - Fonte: Flyway

Existe também a funcionalidade de inserir uma migration que será executada sempre, que é chamada de *Repeatable migrations*. Elas são úteis para gerenciar objetos do banco de dados que não podem ser gerenciados com um simples arquivo no sistema de controle de versão, como criar/recriar *views*, *procedures*, *functions* e etc. Para criar

um arquivo desse tipo basta substituir o “*V+número*” no nome do arquivo por “*R*”, como no exemplo: *R__People_view.sql*.

Uma outra opção são as *Undo migrations* que são responsáveis por desfazer o efeito das *migrations* com a mesma versão, ou seja, *undo migration 1*, reverte o efeito da *migration 1*. As *undo migrations* são opcionais. Para criar um arquivo desse tipo basta substituir o “*V*” no nome do arquivo por “*U*”, como no exemplo: *U1__Create_person_table.sql*. Por padrão é possível desfazer somente a última *migration* aplicada, para aplicar uma *undo migration*, é necessário usar o comando “*flyway undo*”. Na próxima vez que o flyway for executado ele voltará a aplicar a *migration* que foi desfeita.

Existe ainda a opção de uma *migration* escrita em Java, que pode ser útil para alterações difíceis de serem feitas em SQL, como por exemplo, alterações BLOB e CLOB ou re-cálculos e formatações complicadas. Para criar uma *Java migration* basta criar a pasta *src/main/java/db/migration*, depois criar uma classe Java, como no exemplo: *V3__Anonymize.java*. O nome da classe obedece o mesmo padrão de nomenclatura dos arquivos SQL, assim como a mesma ordem. A classe tem de implementar a interface *JdbcMigration*, como no exemplo mostrado na figura 4.

```
package db.migration;

import org.flywaydb.core.api.migration.jdbc.JdbcMigration;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;

public class V3__Anonymize implements JdbcMigration {
    public void migrate(Connection connection) throws Exception {
        try (Statement select = connection.createStatement()) {
            try (ResultSet rows = select.executeQuery("SELECT id FROM person ORDE
R BY id")) {
                while (rows.next()) {
                    int id = rows.getInt(1);
                    String anonymizedName = "Anonymous" + id;
                    try (Statement update = connection.createStatement()) {
                        update.execute("UPDATE person SET name='" + anonymizedNam
e + "' WHERE id=" + id);
                    }
                }
            }
        }
    }
}
```

Figura 4. Migration em Java - Fonte: Flyway

3.2. LIQUIBASE

O funcionamento do Liquibase consiste em criar um arquivo chamado *changelog*, que é onde as alterações de banco de dados são listadas. Este arquivo pode ser escrito em XML, SQL, YAML e JSON. LIQUIBASE (2018). Para iniciar usando XML basta adicionar um arquivo XML vazio, como o mostrado na figura 5.

```
<?xml version="1.0" encoding="UTF-8"?>

<databaseChangeLog
  xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
    http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.1.xsd">

</databaseChangeLog>
```

Figura 5. Criando a arquivo changelog - Fonte: Liquibase

O próximo passo é adicionar um *changeset*, que nada mais é que uma alteração no banco de dados. Cada *changeset* é identificado através dos atributos “*id*” e “*author*” estes atributos juntamente com o nome e o pacote do arquivo *changelog* identificam unicamente a mudança, com intuito de ordenar as mudanças a serem aplicadas no banco dados. O atributo *author* serve para diminuir problemas em ambientes onde existem muitos desenvolvedores trabalhando juntos, onde uma alteração identificada somente pelo *id* seria facilmente duplicada. Um exemplo de arquivo *changeset* pode ser visto na figura 6.

```
<?xml version="1.0" encoding="UTF-8"?>

<databaseChangeLog
  xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
    http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.1.xsd">

  <changeSet id="1" author="bob">
    <createTable tableName="department">
      <column name="id" type="int">
        <constraints primaryKey="true" nullable="false"/>
      </column>
      <column name="name" type="varchar(50)">
        <constraints nullable="false"/>
      </column>
      <column name="active" type="boolean" defaultValueBoolean="true"/>
    </createTable>
  </changeSet>

</databaseChangeLog>
```

Figura 6. Criando a arquivo changeset - Fonte: Liquibase

Existem várias formas de executar o liquibase, dentre elas linha de comando, Ant Maven, Spring, *Server Listener* e CDI. Na figura 7 podemos ver um exemplo da execução por linha de comando.

```
liquibase --driver=com.mysql.jdbc.Driver \  
--classpath=/path/to/classes \  
--changeLogFile=com/example/db.changelog.xml \  
--url="jdbc:mysql://localhost/example" \  
--username=user \  
--password=asdf \  
migrate
```

Figura 7. Executando o liquibase por linha de comando - Fonte: Liquibase

Ao verificar o banco de dados nesse ponto, poderemos ver que existe uma tabela chamada “department”. Outras duas tabelas também foram criadas: “*databasechangelog*” e “*databasechangeloglock*”. A tabela *databasechangelog* possui lista dos *changesets* ou alterações que já foram executados. A tabela *databasechangeloglock* é usada para garantir de duas máquinas não tentem alterar uma mesma tabela ao mesmo tempo.

O Liquibase preza que com o mesmo mapeamento, ou seja, o mesmo arquivo *changelog* e seus *changesets*, possa ser configurado a usar vários SGDBs. Para isso ele é o responsável de gerar o SQL que será executado no banco de dados, observando as particularidades de cada SGDB.

4. ESTUDO DE CASO

Como estudo de caso, foi implementado um pequeno sistema de cadastro de contas a pagar e receber usando as ferramentas Flyway e Liquibase. O objetivo será mostrar as diferenças básicas das ferramentas em termos de uso no dia a dia e uma limitação da ferramenta Liquibase. O foco será na ajuda de tomada de decisão de uso entre uma das ferramentas, dado que as duas são bastante similares em termos do que fazem.

As diferenças que serão mostradas no estudo de caso são as mesmas que segundo KÖBLER (2018) são as únicas grandes diferenças que valem a pena mencionar, no restante, ambas as ferramentas podem ser consideradas iguais pois suas

diferenças são apenas marginais em termos de recursos e conceitos, sendo assim a escolha fica por conta da preferência da equipe com base nas diferenças que serão apresentadas.

A primeira parte do sistema usado para estudo de caso, será a implementação de um simples CRUD de contas, posteriormente, a estrutura do CRUD será alterada e onde haverão as tabelas conta e prestação. Nesses dois momentos os frameworks serão os responsáveis por fazer as alterações no banco de dados. Não serão feitas mais alterações no banco de dados pois apenas com essas duas já é possível fazer uma análise das diferenças das ferramentas que serão relevantes para uma tomada de decisão de uso.

A seguir serão descritos detalhes técnicos para execução e implementação do referido sistema que foi implementado.

4.1. AMBIENTE

Para execução do projeto é necessário ter instaladas e configuradas as seguintes ferramentas:

- Java 1.8 ou superior
- Maven 3.5.3 ou superior
- Cliente GIT 2.17.0 ou superior
- PostgreSQL 9.6 ou superior
- NetBeans IDE 8.2 ou superior

A implementação do sistema de teste citado acima será feita usando o framework Spring Boot, que minimiza o tempo gasto com configurações na criação de aplicações baseadas no framework Spring. Porém detalhes do funcionamento desse framework não serão abordados, já que o objetivo é mostrar o funcionamento dos frameworks Flyway e Liquibase.

4.2. IMPLEMENTAÇÃO COM FLYWAY

A parte relativa ao Flyway começa em configurá-lo no projeto. O que será feito através do Maven, para isso basta adicionar um trecho de código no arquivo *pom.xml*.

Como pode ser visto na figura 8.

```
<dependency>
  <groupId>org.flywaydb</groupId>
  <artifactId>flyway-core</artifactId>
  <version>5.1.1</version>
</dependency>
```

Figura 8. Configuração do Flyway no arquivo pom.xml - Fonte: Do autor

Após configurado, basta criar a estrutura de pastas onde ficam os arquivos com as instruções SQL com as alterações do banco de dados. Por padrão esses arquivos ficam em: *src/main/resources/db/migration*. Após isso, basta adicionar os arquivos SQL nesta pasta, obedecendo o padrão de nomenclatura estabelecida. Vamos adicionar nossa primeira alteração do banco de dados, que é a criação da tabela conta. O conteúdo do arquivo pode ser visto na figura 9.

```
CREATE TABLE conta
(
  id_conta          BIGSERIAL NOT NULL,
  descricao         CHARACTER varying(255) NOT NULL,
  tipo              CHARACTER varying(7) NOT NULL,
  data_vencimento  DATE NOT NULL,
  valor             NUMERIC(21, 2),
  CONSTRAINT conta_pkey PRIMARY KEY (id_conta)
);
```

Figura 9. Arquivo V1_ Criando tabela conta.sql - Fonte: Do autor

Com o arquivo criado, ao executar o projeto o Flyway irá executá-lo e então criar a tabela *conta*. Como é a primeira execução do Flyway ele irá criar sua tabela de controle, que se chama *flyway_schema_history*. A figura 10 mostra o estado do banco de dados nesse momento.

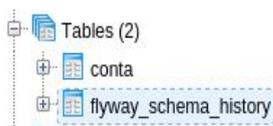


Figura 10. Tabelas criadas pelo Flyway- Fonte: Do autor

Após a implementação do CRUD de contas e da criação da respectiva tabela com a ajuda do Flyway basta adicionar todas as alterações ao Sistema de Controle de Versão. Como podemos ver na figura 11.

Files changed (14)

| | | | |
|-----|----|---|--|
| +5 | -0 | M | pom.xml |
| +92 | -0 | A | src/main/java/br/com/tcc/ismael/controllers/ContaController.java |
| +49 | -0 | A | src/main/java/br/com/tcc/ismael/dao/ContaDao.java |
| +60 | -0 | A | src/main/java/br/com/tcc/ismael/dao/PaginatorQueryHelper.java |
| +75 | -0 | A | src/main/java/br/com/tcc/ismael/modelo/Conta.java |
| +37 | -0 | A | src/main/java/br/com/tcc/ismael/modelo/PaginatedList.java |
| +16 | -0 | A | src/main/java/br/com/tcc/ismael/modelo/TipoConta.java |
| +10 | -0 | A | src/main/resources/db/migration/V1__criando tabela conta.sql |
| +86 | -0 | A | src/main/webapp/WEB-INF/tags/template/admin.tag |
| +24 | -0 | A | src/main/webapp/WEB-INF/tags/template/paginationComponent.tag |
| +15 | -0 | A | src/main/webapp/WEB-INF/views/conta/form-add.jsp |
| +42 | -0 | A | src/main/webapp/WEB-INF/views/conta/form-inputs.jsp |
| +16 | -0 | A | src/main/webapp/WEB-INF/views/conta/form-update.jsp |
| +48 | -0 | A | src/main/webapp/WEB-INF/views/conta/list.jsp |

Figura 11. Arquivos enviados ao Sistema de Controle de Versão - Fonte: Do autor

Com intenção de simular as mudanças na regra de negócio que acontecem em sistema real, a tabela de contas sofrerá a adição da funcionalidade de prestações. A alteração consiste em criar a tabela de prestações, percorrer as contas existentes criando uma prestação para cada conta e excluir os dados referentes a data de vencimento e valor da tabela *conta*. O conteúdo do arquivo *V2__criando tabela prestacao.sql* pode ser visto na figura 12.

```
CREATE TABLE prestacao
(
    id_prestacao BIGSERIAL NOT NULL,
    id_conta BIGSERIAL NOT NULL,
    data_vencimento DATE NOT NULL,
    valor NUMERIC(21, 6),
    CONSTRAINT prestacao_pkey PRIMARY KEY (id_prestacao),
    CONSTRAINT conta_fkey FOREIGN KEY (id_conta) REFERENCES conta (id_conta)
);

DO $$DECLARE
conta RECORD;
BEGIN
FOR conta IN
(SELECT
    id_conta id_conta,
    data_vencimento data_vencimento,
    valor valor
FROM conta)
LOOP
INSERT INTO prestacao(id_conta, data_vencimento, valor)
VALUES ( conta.id_conta, conta.data_vencimento, conta.valor);
END LOOP;
END$$;

ALTER TABLE conta
DROP COLUMN data_vencimento,
DROP COLUMN valor;
```

Figura 12. Arquivo *V2__criando tabela prestacao.sql* - Fonte: Do autor

Após a execução do Flyway a tabela *flyway_schema_history* terá as informações referentes às duas alterações realizadas, como pode ser visto na figura 13.

| installed_rank | version | description | type | script | checksum | installed_by | installed_on | execution_time | success |
|----------------|-----------|--------------------------|--------|---------------------------------|------------|---------------|-----------------------------|----------------|---------|
| integer | character | character varying (200) | charac | character varying (1000) | integer | character var | timestamp without time zone | integer | boolean |
| 1 | 1 | criando tabela conta | SQL | V1_criando tabela conta.sql | -547873915 | postgres | 2018-06-11 08:39:13.218898 | 9 | true |
| 2 | 2 | criando tabela prestacao | SQL | V2_criando tabela prestacao.sql | -69104622 | postgres | 2018-06-11 08:39:13.245785 | 11 | true |

Figura 13. Conteúdo da tabela *flyway_schema_history* - Fonte: Do autor

4.3. IMPLEMENTAÇÃO COM LIQUIBASE

Para começar basta adicionar a configuração indicando qual o caminho do arquivo *changelog*, que é onde serão inseridas as alterações no banco de dados. Para isso é preciso adicionar a linha: *liquibase.change-log=classpath:db/liquibase-changelog.xml* no arquivo *application.properties*. Também é necessário substituir a configuração do Flyway pela configuração do Liquibase. Para isso basta alterar o arquivo *pom.xml*, como pode ser visto na figura 14.

```
<dependency>
  <groupId>org.liquibase</groupId>
  <artifactId>liquibase-core</artifactId>
</dependency>
```

Figura 14. Configuração do Liquibase no arquivo *pom.xml* - Fonte: Do autor

Feito isso basta criar o arquivo *changelog*, obedecendo o caminho e nome configurados anteriormente. Em seguida basta adicionar nele a primeira alteração, que é a criação da tabela *conta*, conforme a figura 15.

```
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog
  xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
    http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.1.xsd">
  <changeSet id="1" author="ismael">
    <createTable tableName="conta">
      <column name="id_conta" type="bigserial">
        <constraints primaryKey="true" nullable="false"/>
      </column>
      <column name="descricao" type="varchar(255)">
        <constraints nullable="false"/>
      </column>
      <column name="tipo" type="varchar(7)">
        <constraints nullable="false"/>
      </column>
      <column name="data_vencimento" type="date">
        <constraints nullable="false"/>
      </column>
      <column name="valor" type="numeric(21,2)">
        <constraints nullable="true"/>
      </column>
    </createTable>
  </changeSet>
</databaseChangeLog>
```

Figura 15. Criação da tabela conta em liquibase-changelog.xml - Fonte: Do autor

A criação do XML implica em inserir as tags conforme a documentação da ferramenta, onde uma tag ou conjunto delas compõem um comando SQL.

Ao executar a aplicação após a inserção da primeira mudança no Liquibase o mesmo irá criar a tabela conta, bem como suas tabelas de controle como pode ser visto na figura 16.



Figura 16. Tabelas criadas pelo Liquibase - Fonte: Do autor

Como foi feito com o Flyway será feita uma alteração no banco que consiste em criar a tabela de prestações, percorrer as contas existentes criando uma prestação para cada conta e excluir os dados referentes a data de vencimento e valor da tabela conta. Para isso vamos adicionar *changesets* ao arquivo *changelog* conforme pode ser visto nas figuras 17 e 18.

```
</changeSet>
<changeSet id="2" author="ismael">
  <createTable tableName="prestacao">
    <column name="id_prestacao" type="bigserial">
      <constraints primaryKey="true" nullable="false"/>
    </column>
    <column name="id_conta" type="bigserial">
      <constraints nullable="false"
        foreignKeyName="conta_fkey"
        references="conta(id_conta)"/>
    </column>
    <column name="data_vencimento" type="date">
      <constraints nullable="false"/>
    </column>
    <column name="valor" type="numeric(21,2)">
      <constraints nullable="false"/>
    </column>
  </createTable>
</changeSet>
<changeSet id="3" author="ismael">
```

Figura 17. Criando prestação parte 1 - Fonte: Do autor

```

</changeSet>
<changeSet id="3" author="ismael">
  <sql dbms="PostgreSQL"
    endDelimiter=";"
    splitStatements="false"
    stripComments="true">
    DO $$DECLARE
    conta RECORD;

    BEGIN
    FOR conta IN
    (SELECT
    id_conta      id_conta,|
    data_vencimento  data_vencimento,
    valor          valor
    FROM conta)
    LOOP
    INSERT INTO prestacao(id_conta, data_vencimento, valor)
    VALUES ( conta.id_conta, conta.data_vencimento, conta.valor);
    END LOOP;

    END$$;
  </sql>
</changeSet>
<changeSet id="4" author="ismael">
  <dropColumn columnName="data_vencimento"
    tableName="conta"/>
  <dropColumn columnName="valor"
    tableName="conta"/>
</changeSet>
atabaseChangeLog>

```

Figura 18. Criando prestação parte 2 - Fonte: Do autor

Após a execução da criação da tabela *prestacao* e alteração da tabela *conta* pelo Liquibase, a sua tabela de controle *databasechangelog* estará como mostrado na figura 19.

| Data Output | | | | | | | | | | | | | | |
|-------------|------|-----------|-------------------------|-----------------------------|---------------|-------------------|-----------|-------------------------|-----------|--------|-----------|-----------|-----------|-------------------|
| | id | author | filename | dateexecuted | orderexecuted | exectype | md5sum | description | comment | tag | liquibase | contexts | label | deployment_id |
| | char | character | character varying (255) | timestamp without time zone | integer | character varying | character | character varying (255) | character | char | character | character | character | character varying |
| 1 | 1 | ismael | classpath:db/liquib... | 2018-06-22 23:49... | 1 | EXECUTED | 7:fb3... | createTable tableNa... | | [null] | 3.5.3 | [null] | [null] | 9722159779 |
| 2 | 2 | ismael | classpath:db/liquib... | 2018-06-23 18:24... | 2 | EXECUTED | 7:2e7... | createTable tableNa... | | [null] | 3.5.3 | [null] | [null] | 9789061551 |
| 3 | 3 | ismael | classpath:db/liquib... | 2018-06-23 18:24... | 3 | EXECUTED | 7:9e3... | sql | | [null] | 3.5.3 | [null] | [null] | 9789061551 |
| 4 | 4 | ismael | classpath:db/liquib... | 2018-06-23 18:28... | 4 | EXECUTED | 7:bd5... | dropColumn column... | | [null] | 3.5.3 | [null] | [null] | 9789312586 |

Figura 19. Conteúdo da tabela *databasechangelog* - Fonte: Do autor

4.4. CARACTERÍSTICAS E DIFERENÇAS ENTRE FLYWAY E LIQUIBASE

Os Dois *Frameworks* são bastante parecidos em termos de problemas que resolvem.

- Formalizam um meio de armazenar as alterações do banco de dados por todos os membros da equipe, fazendo com que estejam versionadas junto ao código. Isso

facilita a integração de código entre os membros da equipe, e também facilita o monitoramento as alterações do banco de dados.

- Comparam a versão do banco de dados usado com a versão que está versionada junto ao código fonte e o atualizam quando necessário.
- Armazenam no banco de dados um histórico de todas as alterações realizadas, com informação de qual foi a alteração, por quem foi feita e quando foi aplicada.

Eles também são parecidos em termos de interesse por pesquisa no google. Como pode ser visto na figura 20. O que indica um número de usuários similar.

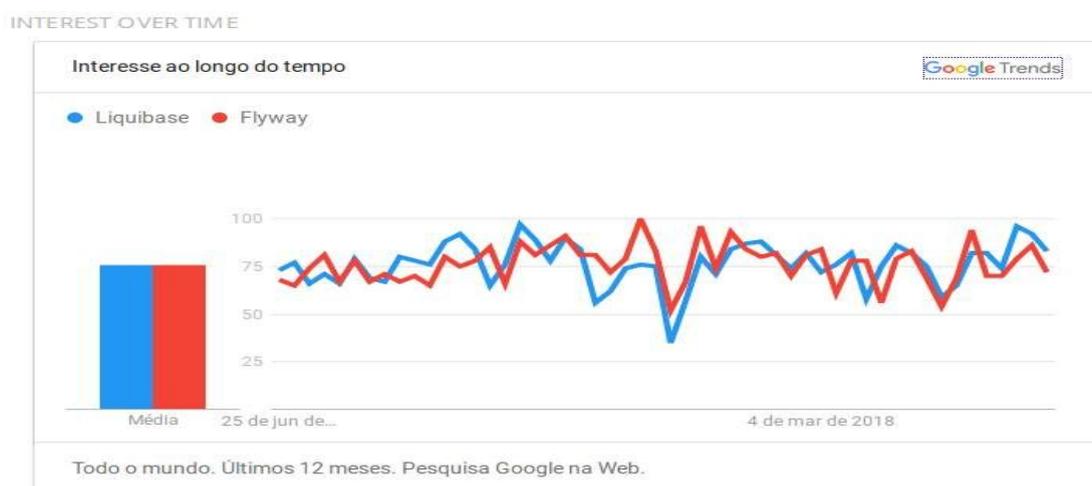


Figura 20. gráfico estatísticas de pesquisas no Google - Fonte: Google Trends

Em termos de diferenciais, basicamente estão na forma como as alterações do banco de dados são escritas.

- No Liquibase as alterações podem ser feitas nos formatos XML, SQL, YAML e JSON onde são chamadas de *changesets*. As *changesets* podem estar divididas em vários arquivos, ou todas em um mesmo. Posteriormente elas serão traduzidas em SQL que será executado no banco de dados.
 - O lado positivo disso é que a mesma *changeset* deveria funcionar em todos os SGDBs suportados. Porém isso encontra um problema, pois nem todas as operações possíveis em um SGDB são suportadas de forma genérica pelo liquibase, ou seja, não poderão ser traduzidas para outros SGDBs. Um exemplo desse problema aconteceu na segunda mudança

realizada no sistemas de Contas, onde uma operação SQL teve de ser mapeada com a tag `<sql>` que é a forma de se inserir uma alteração que não pode ser mapeada de forma genérica no Liquibase.

- Quanto à flexibilidade na disposição das *changesets* em um ou mais arquivos. Se não existir uma padronização definida pela equipe, poderá dificultar na manutenção e no desenvolvimento. Pois poderia ficar difícil rastrear qual foi a última alteração ou uma sequência de alterações, devido ao fato de poderem estar espalhadas por diferentes arquivos.
- No Flyway as alterações são feitas através de código SQL diretamente, que serão dispostos em arquivos texto com extensão *.sql* seguindo um padrão de nomenclatura que determina a ordem de execução das mesmas. As mudanças são chamadas de *migrations*. Outra forma possível de se escrever uma *migration* é através de código Java.
 - Dessa forma, as alterações podem ser feitas quase sem necessidade de esforço adicional, já que SQL é a forma convencional de fazer alterações. Mas por outro lado o uso dos mesmos arquivos SQL em um SGDB diferente não será possível sem necessidade de refatoração.

5. CONSIDERAÇÕES FINAIS

O gerenciamento de mudanças em bancos de dados enfrenta problemas com falta de definição de um processo e necessidade de muito trabalho de análise manual. Ferramentas gerenciadoras de mudanças de bancos de dados mostraram ser um passo importante na solução da falta de um processo de gerência dessas mudanças, propiciando a automação da aplicação das referidas mudanças, o que evita o trabalho manual, poupando tempo no dia a dia de desenvolvedores e DBAs.

O artigo apresentou as ferramentas Flyway e Liquibase que são úteis na automação do gerenciamento de mudanças de bancos de dados. Também apresentou um estudo de caso mostrando as principais diferenças destas ferramentas, de forma a ser uma base para uma escolha entre elas.

Ambas ferramentas conseguem automatizar a gerência de mudanças em bancos de dados e também mostraram permitir que as alterações do código e do banco de dados fiquem juntas no Sistema de Controle de Versão. Sendo assim pode-se dizer que as duas são opções viáveis para minimizar os problemas encontrados na gerência de mudanças em banco de dados.

Trabalhos futuros podem ser desenvolvidos a cerca de analisar quantitativamente o tempo poupado com o uso das ferramentas em uma equipe de desenvolvimento. Também pode ser estudado esse problema da gerência de mudanças de bancos de dados dentro do universo de outras linguagens de programação.

REFERÊNCIAS

APOLINÁRIO, DANIEL RODRIGO DE FREITAS; QUEIRÓS, LEONARDO RIBEIRO; CRUZ, SÉRGIO APARECIDO BRAGA DA. **VERSIONAMENTO DE BANCO DE DADOS COM A FERRAMENTA LIQUIBASE: APLICAÇÃO AO MÓDULO WEB DO SISTEMA DE INFORMAÇÃO DE EXPERIMENTOS DA EMBRAPA (SIExp)**. 2015. COMUNICADO TÉCNICO. DISPONÍVEL EM: <[HTTPS://AINFO.CNPITIA.EMBRAPA.BR/DIGITAL/BITSTREAM/ITEM/138111/1/COMTEC118.PDF](https://ainfo.cnptia.embrapa.br/digital/bitstream/item/138111/1/ComTec118.pdf)>. ACESSO EM 30 DE JUNHO DE 2018

DATICAL. **THE ROI OF AUTOMATING DATABASE DEPLOYMENTS**. DISPONÍVEL EM: <[HTTP://PAGES.DATICAL.COM/RS/522-INH-443/IMAGES/WHITE-PAPER-THE-ROI-OF-AUTOMATING-DATABASE-DEPLOYMENTS.PDF](http://pages.datical.com/rs/522-INH-443/images/white-paper-the-roi-of-automating-database-deployments.pdf)>. ACESSO EM: 16 DE ABRIL DE 2018

FLYWAY. **FLYWAY BY BOXFUSE - DATABASE MIGRATIONS MADE EASE**. DISPONÍVEL EM: <[HTTPS://FLYWAYDB.ORG/](https://flywaydb.org/)>. ACESSO EM: 11 DE ABRIL DE 2018

GOOGLE. **LIQUIBASE, FLYWAY -PESQUISAR - GOOGLE TRENDS**. DISPONÍVEL EM: <[HTTPS://TRENDS.GOOGLE.COM/TRENDS/EXPLORE?DATE=TODAY%2012-M,TODAY%2012-M&GEO=,&q=LIQUIBASE,FLYWAY](https://trends.google.com/trends/explore?date=today%2012-m,today%2012-m&geo=&q=liquibase,flyway)>. ACESSO EM 29 DE JUNHO DE 2018

JOTZ, GUSTAVO. **EVOLUÇÃO DE BANCO DE DADOS NA INTEGRAÇÃO CONTÍNUA.** DISPONÍVEL EM:
<[HTTP://CWISOFTWARE.GITHUB.IO/DROPS/EVOLUCAO-DE-BANCO-DE-DADOS-NA-INTEGRACAO-CONTI
NUA](http://cwisoftware.github.io/drops/evolucao-de-banco-de-dados-na-integracao-conti-nua)>. ACESSO EM: 16 DE ABRIL DE 2018

KÖBLER, NIKO. **CONTINUOUS DATABASE MIGRATION WITH LIQUIBASE AND FLYWAY.**
DISPONÍVEL EM:
<[HTTP://WWW.H-ONLINE.COM/DEVELOPER/FEATURES/CONTINUOUS-DATABASE-MIGRATION-WITH-LI
QUIBASE-AND-FLYWAY-1860080.HTML](http://www.h-online.com/developer/features/continuous-database-migration-with-liquibase-and-flyway-1860080.html)>. ACESSO EM: 21 DE ABRIL DE 2018

LIQUIBASE. **LIQUIBASE - DATABASE REFACTORING.** DISPONÍVEL EM:
<[HTTP://WWW.LIQUIBASE.ORG/INDEX.HTML](http://www.liquibase.org/index.html)>. ACESSO EM: 09 DE ABRIL DE 2018

SOMMERVILLE, IAN. **ENGENHARIA DE SOFTWARE.** PEARSON ADDISON WESLEY, 9ª ED. SÃO
PAULO, 2011.

ZIMERMANN, JOSÉ ALBERTO; SOUSA, THIAGO CARVALHO; SPÍNOLA, RODRIGO
OLIVEIRA. **GERENCIANDO MUDANÇAS A PARTIR DE REQUISITOS.** DISPONÍVEL EM:
<[HTTP://WWW.DEVMEDIA.COM.BR/GERENCIANDO-MUDANCAS-A-PARTIR-DE-REQUISITOS-REVISTA-EN
GENHARIA-DE-SOFTWARE-MAGAZINE-44/23331](http://www.devmedia.com.br/gerenciando-mudancas-a-partir-de-requisitos-revista-engenharia-de-software-magazine-44/23331)>. ACESSO EM 11 DE ABRIL DE 2018