IGOR AUGUSTO COSTA NUNES

IMPLEMENTAÇÃO DE EVOLUTIONS PARA A MANUTENÇÃO DE BANCO DE DADOS DE SISTEMAS AMBIENTAIS

LAVRAS - MG

2019

IGOR AUGUSTO COSTA NUNES

IMPLEMENTAÇÃO DE EVOLUTIONS PARA A MANUTENÇÃO DE BANCO DE DADOS DE SISTEMAS AMBIENTAIS

Relatório de Estágio apresentado à Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação, para a obtenção do título de Bacharel.

Prof. Dr. Ramon Gomes Costa
Orientador

LAVRAS - MG
2019

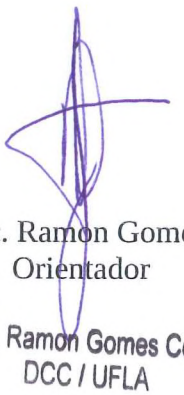
IGOR AUGUSTO COSTA NUNES

IMPLEMENTAÇÃO DE EVOLUTIONS PARA A MANUTENÇÃO DE BANCO DE DADOS DE SISTEMAS AMBIENTAIS

Relatório de estágio apresentado à Universidade Federal de Lavras como parte das exigências do Curso de Ciência da Computação para obtenção do título de Bacharèl em Ciência da Computação.

APROVADA em 12 de Novembro de 2019.

Prof. DSc. Ramon Gomes Costa	DCC/UFLA
Prof. DSc. Janderson Rodrigo de Oliveira	DCC/UFLA
Dalberth Sulivan Mathias	LEMAF/UFLA



Prof. D.Sc. Ramon Gomes Costa
Orientador

Prof. Ramon Gomes Costa
DCC / UFLA

LAVRAS – MG
2019

Dedico este trabalho aos meus pais Wellington Custódio e Luzia Aparecida, aos meus tios Vera, Wilson, Maria do Carmo, Flávio, Celio e Celia e amigos Maria Maciel e Sebastião.

AGRADECIMENTOS

Agradeço primeiramente a Deus, por me ajudado a enfrentar meus obstáculos durante toda esta caminhada. Agradeço a minha família estar sempre presente me auxiliando nos momentos mais difíceis e prestando toda força e apoio. Agradeço meus amigos por caminharem e participarem de toda esta trajetória junto a mim. Agradeço meus professores por todos os ensinamentos e experiências transmitidas e demonstradas.Por fim, porém não menos importante, agradeço aos meus companheiros de trabalho por sempre estarem apoiando e compartilhando novos conhecimentos.

A todos vocês meu muito obrigado.

RESUMO

Com o aumento significativo da quantidade de sistemas Web e com o intuito de auxiliar o governo a manter a regularidade ambiental, há a necessidade de armazenamento eficiente de informação e, consequentemente, de manutenção da integridade dos dados. Neste trabalho, são apresentadas as atividades realizadas durante o estágio na FUNDECC (Fundação de Desenvolvimento Científico e Cultural), em atividades realizadas no LEMAF (Laboratório de Estudos e Projetos em Manejo Florestal). Com atividades de administração de banco de dados em projetos governamentais com a utilização de metodologias ágeis para a resolução de problemas relatados pelos clientes, foram feitas implementações de *Evolutions* para a manutenção de banco de dados juntamente com os sistemas Web, para manter os dados seguros e consistentes. A falta de espaço em servidores, diagramas de banco de dados inexistentes e por vezes feitos de forma manual e as formas de aplicar as *Evolutions* foram pontos de melhorias considerados; melhorias propostas no trabalho são essenciais para o bom funcionamento dos sistemas e convívio com os demais colaboradores.

Palavras-chave: banco de dados; administração de banco de dados; metodologia ágil; ambiental; evolution

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

A graduação é um momento único na vida de um estudante. Independente do curso que esteja realizando, a vida acadêmica tem de ser proveitosa, haja visto que além da graduação há a necessidade de complemento às atividades acadêmicas.

Então, com o intuito de complementar os conhecimentos adquiridos durante o curso de graduação em Ciência da Computação cursado na Universidade Federal de Lavras (UFLA), foram realizadas atividades de estágio na Fundação de Desenvolvimento Científico e Cultural (FUNDECC), em um laboratório denominado Laboratório de Estudos e Projetos em Manejo Florestal (LEMAF), que desenvolve sistemas governamentais para os diversos âmbitos e vertentes ambientais.

O LEMAF, fundado em 2004, está inserido no Departamento de Ciências Florestais da Universidade Federal de Lavras (DCF/UFLA), que tem como objetivo efetuar pesquisa, ensino e extensão. O setor de Tecnologia da Informação passou a existir em meados de 2006. Os primeiros profissionais desse setor somente davam suporte à infraestrutura computacional do ambiente.

As principais áreas de atuação do LEMAF são: geoprocessamento; economia florestal; sistema de informação; software para gestão ambiental; sistemas de informação geográficas; sensoriamento remoto; manejo; e inventário florestal.

Em se tratando de questões ambientais, o LEMAF possui em sua maioria produtos na qual ajuda os governos de muitos estados a ter informação sobre os recursos ambientais que são utilizados não só pela população mas também por grandes empresas do país. Assim, através dos

sistemas produzidos em parceria com o LEMAF, o governo tem controle em não permitir a escassês de recursos ambientais como recursos hídricos e áreas desmatadas, por exemplo. Dentre os produtos podem ser citados os sistemas: Sistema de Outorga de Água do Rio Grande do Sul (SIOUT-RS); Sistema de Gestão de Recursos Hídricos do Pará (SIGERH-PA); Adequação Ambiental; Sistema Integrado de Monitoramento e Licenciamento Ambiental (SIMLAM); e Sistema de Comercialização e Transporte de Produtos Florestais (SISFLORA). Estes, foram projetos dos quais o estagiário participou.

O objetivo desse trabalho é apresentar as atividades exercidas no período de sete meses de estágio, no período de vigência de 07/01/2019 a 31/07/2019, com carga horária de 30 horas semanais, totalizando 360 horas, dentro da empresa LEMAF, detalhando as ferramentas utilizadas e o cenário de atuação da empresa, garantindo a integridade e segurança das informações armazenadas na base de dados dos sistemas. Para tal, esse trabalho está dividido em: Contextualização do mercado e apresentação da empresa; Tecnologias utilizadas; Atividades desenvolvidas; Implementação de Evolutions para a manutenção de Banco de Dados de Sistemas Ambientais; e Considerações finais.

2 CONTEXTUALIZAÇÃO DO MERCADO E APRESENTAÇÃO DA EMPRESA

Este capítulo fornece uma descrição geral da empresa onde o estágio foi realizado, incluindo informações sobre o histórico, descrição física, plataforma de produtos, mercados atendidos pela empresa, número de funcionários, entre outros.

2.1 LEMAF

O LEMAF, fundado em 2004, está inserido no DCF/UFLA, que tem como objetivo efetuar pesquisa, ensino e extensão. O LEMAF conduz diversos projetos em parceria e convênio, com órgãos estaduais e federais, bem como com a iniciativa privada. Estes estudos subsidiam o crescimento, desenvolvimento e aperfeiçoamento do ensino.

O setor de Tecnologia da Informação (TI), passou a existir em meados de 2006. Os primeiros profissionais desse setor somente davam suporte à infraestrutura computacional do ambiente. O Inventário Florestal do Estado de Minas Gerais, desenvolvido para a Secretaria de Estado de Meio Ambiente, foi o primeiro projeto com a participação de profissionais de TI. Em decorrência do sucesso do projeto, a Secretaria do Estado de Minas Gerais (SEMAD) deu continuidade a sua parceria com o LEMAF, dando início ao desenvolvimento de sistemas de informação geográfica, o que possibilitou a criação do setor de TI na instituição.

2.2 Mercado

As principais áreas de atuação do LEMAF são: geoprocessamento; economia florestal; sistemas de informação; software para gestão ambiental; sistemas de informação geográficas; sensoriamento remoto; manejo; e inventário florestal.

O LEMAF visa organizar e fortalecer as estruturas geradoras de conhecimento, tecnologias e prestação de serviços, gerando negócios com valor agregado para o setor de gestão ambiental.

2.3 Participações em Projetos

Durante o estágio, depois de um treinamento para se adequar às tecnologias e ferramentas usadas pela empresa, o estagiário participou de 5 projetos que estavam em desenvolvimento, referentes a 2 estados: o estado do Rio Grande do Sul; e o estado do Pará. Nas subseções a seguir os projetos são descritos.

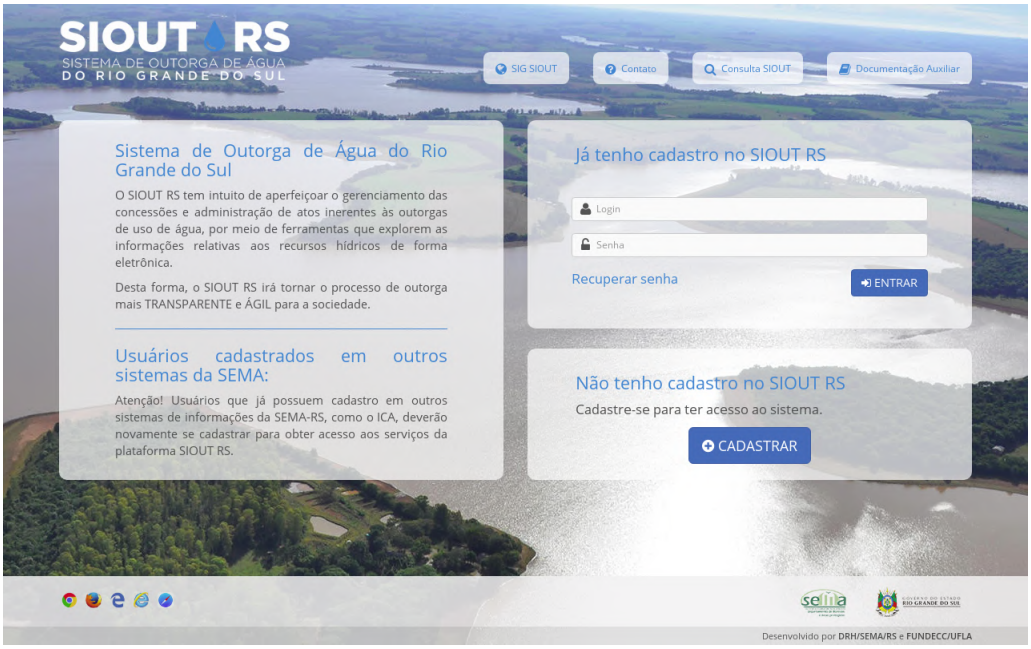
2.3.1 SIOUT-RS

O Sistema de Outorga de Água do Rio Grande do Sul (SIOUT RS) (Figura 2.1) consiste em um conjunto de soluções sistêmicas baseadas em conhecimento para gestão de recursos hídricos e informações climatológicas consolidadas, visando a modernização da gestão integrada dos atos autorizativos de recursos hídricos do Estado do Rio Grande Sul (SIOUT-RS, 2019).

O SIOUT-RS possui, como um de seus objetivos fundamentais, permitir que a Secretaria do Ambiente e Desenvolvimento Sustentável do Rio

Grande do Sul (SEMA-RS) tenha uma visão conjunta da disponibilidade hídrica e possíveis conflitos nos usos da água, possibilitando estabelecer políticas governamentais integradas à Regularização Ambiental do Estado (SIOUT-RS, 2019).

Figura 2.1 – Sistema SIOUT-RS



Fonte: (SIOUT-RS, 2019)

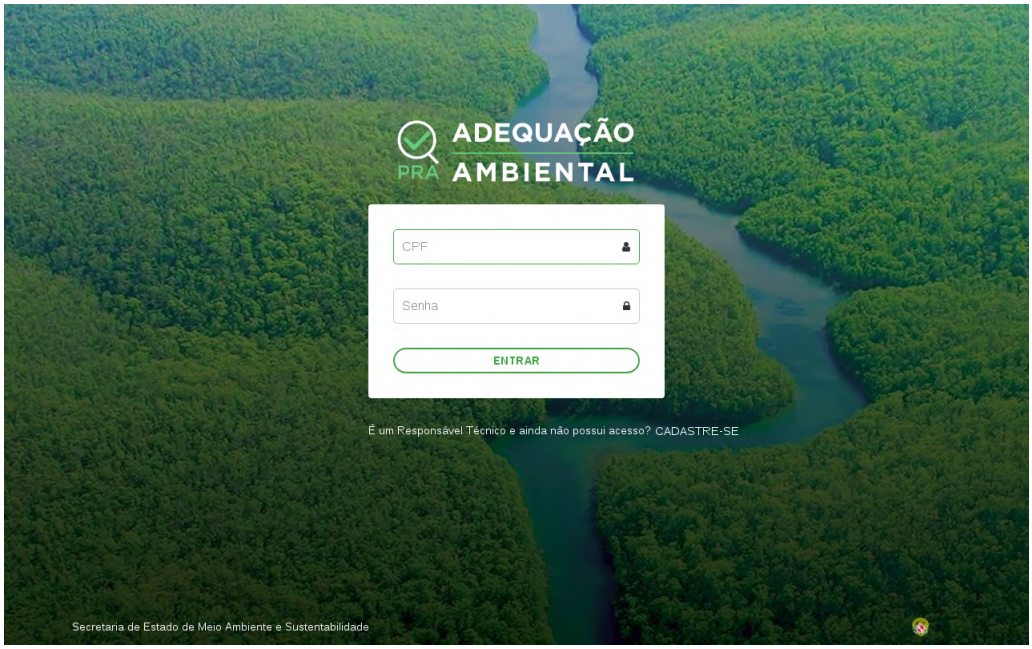
2.3.2 Adequação Ambiental

O Adequação Ambiental (Figura 2.2) é um sistema do estado do Pará que tem como objetivo dar a oportunidade do proprietário com Imóvel Rural (IR) irregular se regularizar no Cadastro Ambiental Rural (CAR[1]) através de um termo de compromisso comprometendo-se a regularizar-se através de implementações de medidas, pré-determinadas, para recupera-

[1] http://www.car.gov.br/#/

ção ambiental do imóvel em débito. Este, também possui o objetivo de aliviar a sobrecarga do sistema de análise do imóvel.

Figura 2.2 – Sistema Adequação Ambiental



Fonte: (AMBIENTAL, 2019)

2.3.3 SIMLAM

O Sistema Integrado de Monitoramento e Licenciamento Ambiental (SIMLAM), possui dois módulos: **Interno** e **Público** (PÚBLICO, 2019).

O SIMLAM - Módulo Interno (SIMLAM Interno) tem como objetivo facilitar a comunicação entre os responsáveis técnicos e a SEMAS, através da disponibilização de todos os roteiros e modelos utilizados na secretaria. Este possibilita o acompanhamento pelo responsável técnico da tramitação e, quando for o caso, pendências do processo de licenciamento ambiental (PÚBLICO, 2019).

O SIMLAM - Módulo Público (SIMLAM Público) (Figura 2.3) tem como objetivo disponibilizar para o público em geral um acompanhamento dos processos e das atividades licenciadas pela SEMAS/PA, com o objetivo de imprimir transparência e eficiência à política ambiental (PÚBLICO, 2019).

Figura 2.3 – Sistema SIMLAM Público

Fonte: (PÚBLICO, 2019)

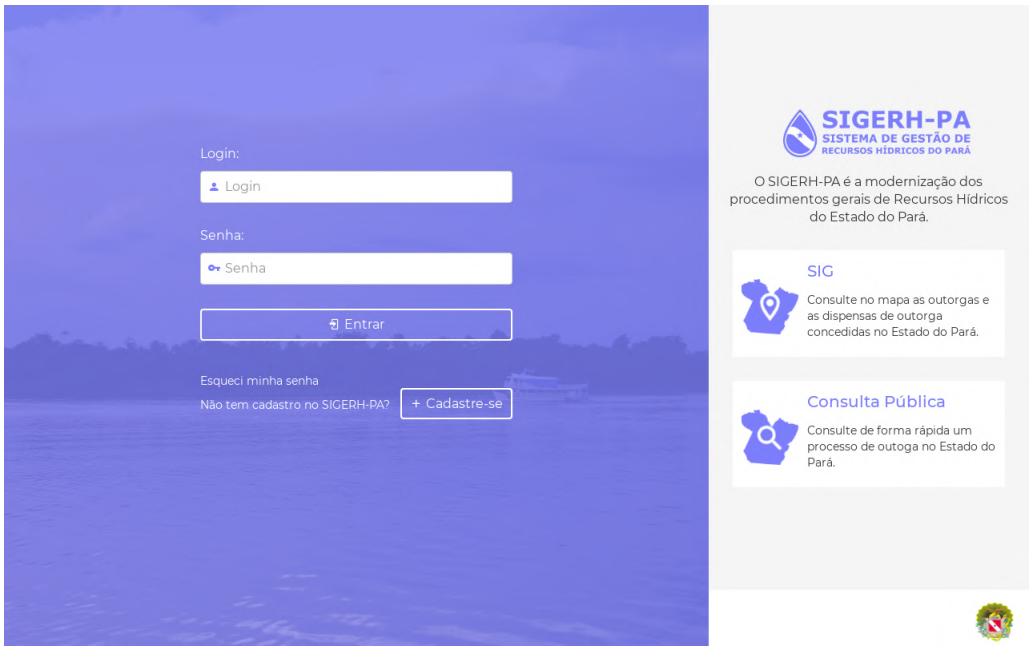
2.3.4 SIGERH-PA

O Sistema de Gestão de Recursos Hídricos do Pará (SIGERH-PA) (Figura 2.4) trata-se de um conjunto de soluções sistêmicas eletrônicas baseadas em conhecimento de gestão de recursos hídricos e informações hidrológicas, hidrometeorológicas e hidrogeológicas consolidadas.

Tem como objetivo a modernização da gestão de recursos hídricos realizada pela SEMAS/PA, com ênfase na modernização geral dos proce-

dimentos que envolvem desde as fases iniciais, de protocolo de processos e recepção de documentos, passando por todas as fases de análise, até a expedição dos atos autorizativos de uso de recursos hídricos (ASCOM, 2017).

Figura 2.4 – Sistema SIGERH-PA



Fonte: (SIGERH-PA, 2019)

Figura 2.5 – Sistema SISFLORA

Fonte: (SISFLORA, 2019)

2.3.5 SISFLORA

O Sistema de Comercialização e Transporte de Produtos Florestais (SISFLORA) (Figura 2.5) integrado ao Sistema de Cadastro de Consumidores de Produtos Florestais (CEPROF) é um sistema que tem como objetivo auxiliar e controlar a comercialização e o transporte de produtos florestais no estado do Pará (SISFLORA, 2019).

3 TECNOLOGIAS UTILIZADAS

Neste capítulo são apresentadas as tecnologias utilizadas durante o período de estágio na empresa LEMAF.

3.1 PostgreSQL

O PostgreSQL[1] é um Sistema Gerenciador de Banco de Dados (SGBD) objeto-relacional de código aberto (CARVALHO, 2017).

O PostgreSQL tem como características:

- "Comandos SQL são consistentes entre si e por padrão;
- É transacional, incluindo mudanças estruturais destrutivas;
- Suporta muitos tipos de dados sofisticados, incluindo *JSON*, *XML*, objetos geométricos, hierarquias, tags, e matrizes. Novos tipos de dados e funções podem ser escritos em *SQL*, *C*, ou linguagens procedurais muito incorporadas, incluindo *Python*, *Perl*, *TCL* e outras;
- Faz uso estratégico de indexação e consulta de otimização para trabalhar com o menor esforço possível" (CARVALHO, 2017).

O PostgreSQL é *Open Source* e gratuito.

Durante o estágio, este SGBD foi a ferramenta base de trabalho. Este era responsável por armazenar as informações da maioria dos projetos para manter a segurança e integridade dos dados. Além disso, por este foram realizadas consultas, criados novos objetos (entidades, *views*, *stored procedures*, funções, etc) através de *evolutions* (que são uma série de *scripts*

[1] https://www.postgresql.org/

SQL que contemplam as configurações do banco de dados), realizar backups, administração de usuários, entre outras atividades. Estas atividades serão descritas em seções posteriores.

3.2 Oracle

De acordo com (GONÇALVES, 2014), o Oracle[2] foi o primeiro SGBD Relacional (SGBDR) comercial. Uma característica deste SGDBR é a integração no mercado da tecnologia da informação, acompanhando as novidades que surgem, como a inclusão de multimídia, CAD/CAM, gráficos, textos, desenhos, fotografias, som e imagem.

O Oracle pode ser instalado em sistemas UNIX, VMS, MVS, HP, Macintosh, Rede Novell, MS-Windows entre outros.

Assim como o PostgreSQL, o SGBD Oracle também foi uma ferramenta base no estágio. Logo, também é uma ferramenta muito importante para o armazenamento de informações de alguns projetos. Porém, todos os projetos desenvolvidos desde o seu início utilizam o PostgreSQL. Os projetos (SIMLAM e SISFLORA), que possuem o Oracle como SGBD, são projetos que já tinham sido desenvolvidos por outras empresas e posteriormente foram entregue ao LEMAF para dar continuidade em seu desenvolvimento. Assim como a aplicação, a base de dados não foi desenvolvida pelo LEMAF. As atividades com relação a essas bases de dados foram: consultas, manutenções de *stored procedures* e funções, criação de *evolutions*, etc.

[2] https://www.oracle.com/br/database/

3.3 Scrum

Scrum[3] é uma metodologia ágil para gestão e planejamento de projetos de software (SCRUM, 2019). No *Scrum*, os projetos são divididos em ciclos, normalmente mensais, chamados de *Sprints*. O *Sprint* possui um *Time Box* na qual possui um conjunto de atividades que serão desenvolvidas durante este tempo.

As funcionalidades que devem ser implementadas no projeto são direcionadas para uma lista denominada *Product Backlog*. No início de cada *Sprint*, faz-se um *Sprint Planning Meeting*, ou seja, uma reunião que tem como objetivo planejar as atividades a serem desenvolvidas, na qual o *Product Owner* prioriza os itens do *Product Backlog* e a equipe seleciona as atividades que ela é capaz de implementar durante o período do *Sprint* que se inicia. As tarefas alocadas em um *Sprint* são transferidas do *Product Backlog* para o *Sprint Backlog*. As atividades que não foram concluídas com êxito, automaticamente serão alocadas na próxima *Sprint* com maior prioridade para serem desenvolvidas.

A cada dia de uma *Sprint*, a equipe faz uma breve reunião, geralmente pela manhã, chamada *Daily Scrum* onde o objetivo é relatar tudo sobre o que foi feito no dia anterior, identificar impedimentos e priorizar o trabalho do dia que se inicia.

Ao final de um *Sprint*, a equipe apresenta as funcionalidades implementadas em uma *Sprint Review Meeting*. Finalmente, faz-se uma *Sprint Retrospective* e a equipe parte para o planejamento do próximo *Sprint*. Assim reinicia-se o ciclo.

[3] https://www.desenvolvimentoagil.com.br/scrum/

Durante o estágio, nas equipes em que o estagiário foi alocado, não se trabalhava em cima de *Sprints*. Utilizava-se a *Daily Scrum* para reunir a equipe e apresentar aos integrantes as atividades que estavam sendo feitas antes dela, informar impedimentos e sugerir soluções, além de indicar se continuará trabalhando nas atividades de antes ou nas que irá desenvolver. A *Sprint Planning Meeting* também foi utilizada para estimar o tempo do desenvolvimento de determinadas atividades. A *Sprint Review Meeting* foi utilizada para revisar as atividades que foram realizadas pela equipe durante o período estipulado. Por fim, foi utilizado a *Sprint Retrospective* na qual realiza um levantamento do que foi feito durante um intervalo de dias estipulado e aponta pontos positivos e pontos a melhorar durante este período.

3.4 Kanban

Kanban é um sistema ágil e visual para controle de produção ou gestão de tarefas (ESPINHA, 2019). O *Kanban* divide-se em três partes:

- **Cartão**: o cartão é a menor parte do *Kanban*. Tem como objetivo descrever uma tarefa ou ação a ser tomada para que o resultado final seja entregue;

- **Colunas**: as colunas representam o *status* no qual o cartão se encontra. Geralmente possui três colunas: "**A Fazer**", "**Em Execução**"e "**Feito**". Essas colunas podem mudar (contendo mais *status*) de acordo com a necessidade da equipe de trabalho, caso o fluxo seja dividido;

- **Quadro**: o quadro é organizado em colunas e cartões. Cada quadro é referente a um *Kanban* mas uma única equipe pode trabalhar em vários quadros.

Durante o estágio, o *Kanban* foi de grande importância para a equipe, pois pôde-se ver todo o fluxo de trabalho e as atividades do projeto de uma maneira visual. Isto facilita a análise do fluxo destas atividades, a evolução do sistema e a verificação da existência de algum gargalo durante o desenvolvimento. Assim pode-se encontrar alternativas na solução do problema para que o fluxo se torne mais ágil.

3.5 Taiga

O Taiga[4] é uma plataforma de gerenciamento de projetos para desenvolvedores, administradores de banco de dados (DBA - *Database Administrator*), *designers* ágeis e gerentes de projetos que desejam uma ferramenta bonita que torne o trabalho realmente agradável.

O Taiga possui várias funções. Algumas são:

- Problemas: é uma função que mantém uma lista dos problemas e erros encontrados nos sistemas, os classifica de acordo com prioridade e os atribui a algum membro da equipe;

- Wiki: é um espaço para a documentação do projeto. É possível editar e enviar o conteúdo para os demais membros da equipe;

- Tarefas: comunica a necessidade de fazer algum trabalho. Os membros da equipe podem criar e definir tarefas para representar o trabalho necessário para realizar uma história do usuário;

[4] https://taiga.io/

- Equipe: Exibe quem e como a equipe está compartilhando as tarefas do projeto e;

- Kanban: Controla as atividades desenvolvidas durante a *Sprint*. Este quadro possui colunas para o controle de atividades como: *"TO DO"*; *"DOING"*; *"DEV DONE"*; *"TO DO TEST"*; *"TEST DOING"*; *"ERROR"*; *"DONE"*; *"IMPEDED"*; *"CANCELED"*; entre outras.

O Taiga para administradores de banco de dados, durante o estágio, foi a ferramenta principal para a visão como um todo do projeto e junto da metodologia ágil. As *tags* de *status* mais utilizadas foram *"TO DO BD"* e *"BD DOING"*, indicando a realização de uma atividade para o DBA, como por exemplo: criar e/ou validar uma *evolution*; realizar um ***backup*** da base dados; entre outras.

3.6 Alfresco

O Alfresco[5] é um sistema de Gestão de conteúdo empresarial (em inglês ECM "Enterprise Content Management"), multiplataforma (Windows e Unix/Linux) de Código Aberto. O Alfresco se propõe a ser uma alternativa para o gerenciamento de documentos, arquivos, colaboração e conteúdos Web (ALFRESCO, 2019).

No LEMAF o Alfresco é utilizado para o gerenciamento de documentos de projetos e seus processos. Logo, toda a documentação técnica, contratual e gerencial aloca-se nesta ferramenta.

Além das ferramentas utilizadas de forma geral, como as mencionadas acima, é no Alfresco que são armazenados os Diagramas modelados

[5] https://www.alfresco.com/

para a criação dos Banco de Dados utilizados em cada projeto. Portanto, a cada atualização na arquitetura da base de dados, deve-se atualizar o diagrama que se encontra nesta ferramenta.

3.7 GitLab

GitLab[6] é um repositório de código e plataforma de desenvolvimento colaborativo *open source* (ROUSE, 2018). O GitLab oferece um local para armazenamento de código *online* e desenvolvimento colaborativo de projetos de *software*. O repositório contém controle de versão para permitir a hospedagem de diferentes cadeias de desenvolvimento e versões.

Durante o estágio, o GitLab foi o repositório tanto das *evolutions* desenvolvidas quanto do projeto inteiro do sistema. Por fornecer controle de versão, todas as *releases* dos sistemas estão nos repositórios do LEMAF, que são privados. Por fim, em se tratando de sistemas para Gorvernos de muitos estados, é de extrema importância que os repositórios não fiquem abertos ao público, garantindo que apenas funcionários e parceiros do LEMAF tenham acesso.

[6] https://about.gitlab.com/

4 ATIVIDADES DESENVOLVIDAS

Nesse capítulo, são apresentadas algumas das atividades desenvolvidas durante o período de estágio no LEMAF. Também é apresentada uma relação entre conceitos aprendidos em sala de aula e o que foi utilizado no estágio.

4.1 Configuração do Ambiente

No momento em que foi pensado sobre desenvolver uma aplicação, surgiram algumas incógnitas. Uma delas é o banco de dados: antes de começar a implementar um banco de dados, deve-se levar em consideração a configuração do ambiente em que este banco de dados estará implantado.

Uma das configurações que devem ser consideradas se encontra no *Hardware*. É necessário analisar qual processador será utilizado, pois um servidor de banco de dados consome bastante processamento, uma vez que há vários serviços sendo executados ao mesmo tempo. Estes serviços não são somente do Sistema Operacional, mas também de serviços do SGBD. A propósito, o processador deve ser multinúcleo. Outro ponto a ser analisado é o espaço em disco: é sabido que para banco de dados, principalmente para aplicações que armazenam muitas informações, o armazenamento em disco é crucial. Denominamos esta etapa como **Projeto Físico**.

Também deve-se fazer escolhas relacionadas ao *Software*. Deve-se definir qual é a melhor opção entre os Sistemas Operacionais (SO) disponíveis para ser utilizado no servidor. Em apenas dois sistemas (SIMLAM e SISFLORA) foram utilizados o SO Windows, pois foram projetos que já tinham sido iniciados por outra empresa e posteriormente se tornou um produto do LEMAF. Já os demais sistemas, que são produtos por completo da

empresa, são utilizados o SO Linux. A equipe de banco de dados (equipe do estagiário) e a equipe da infraestrutura optaram pela utilização de Sistemas Operacionais Linux, pois além de não ter cobrança de licença de *Software*, permite que as empresas economizem com a infraestrutura e possam fornecer conteúdo, aplicativos e serviços para seus clientes (REDAÇÃO, 2019). Esta etapa denominamos como **Projeto Lógico**.

Nesta atividade, o estagiário participou parcialmente junto à equipe de infraestrutura do LEMAF.

4.2 Instalação do SGBD

Com a configuração do ambiente resolvida, é necessário a instalação do SGBD e, independente do sistema operacional, deve-se explorar toda a sua potencialidade. O primeiro ponto a ser analisado é: Qual o SGBD a ser utilizado? Os projetos criados no LEMAF utilizam o SGBD PostgreSQL. Um sistema gerenciador de banco de dados *open source*. A equipe de banco de dados, ao fazer a instalação de um SGBD, usou um tutorial com cada passo a ser seguido para manter o banco de dados bem configurado. Esse tutorial foi desenvolvido por outros DBAs que fizeram parte dos colaboradores do LEMAF. Este documento descreve o que deve ser feito desde a instalação do PostgreSQL até as configurações de variáveis de ambiente. Sempre que é encontrado algo de errado no documento ou algo já ultrapassado, cria-se um novo documento com o tutorial, atualizando-o para que um próximo funcionário não encontre dificuldades para realizar a instalação.

Como deseja-se um ambiente bem definido e configurado, não instala-se o SGBD por meio dos pacotes que apresentam nos gerenciadores de pa-

cotes (considerando o SO Linux) e sim através do código fonte. Assim, pega-se o código fonte, compila-se e instala-se a partir disso. Porém, para instalar desta forma, é preciso algumas bibliotecas solicitadas por dependências de pacotes. Este é um processo massivo, mas realizando a instalação desta maneira tem-se certeza de onde cada item do banco de dados está armazenado.

Em relação à versão do SGBD, sempre se escolhe a versão atual, ou seja, a última lançada, pois há maior garantia de que traga consigo correção de *bugs*, otimização interna para tornar a *query* utilizada pelo DBA mais rápida, outros tipos de consultas que possam facilitar a escrita, entre outros recursos. SGBDs já instalados e em uso em Sistemas Web pré-existentes não são atualizados para uma versão mais nova pois isso requer um levantamento com a equipe de DBA, juntamente com a equipe desenvolvedores e clientes para decidir a necessidade de migrar o SGBD para a versão mais recente. Outro ponto a ser levado em consideração é que este processo não é simples. Migrar um banco de dados inteiro e que já está em funcionamento pode trazer complicações. Por outro lado, as novidades da nova versão, que podem resolver um problema do banco de dados com a versão antiga, não poderão ser utilizadas, tendo que contornar tal problema de outra maneira. Por fim, há prós e contras.

Nesta atividade, o estagiário participou de todas as etapas junto à equipe de infraestrutura para a instalação do SGBD .

4.3 Planejar, Implementar e criar o Banco de Dados

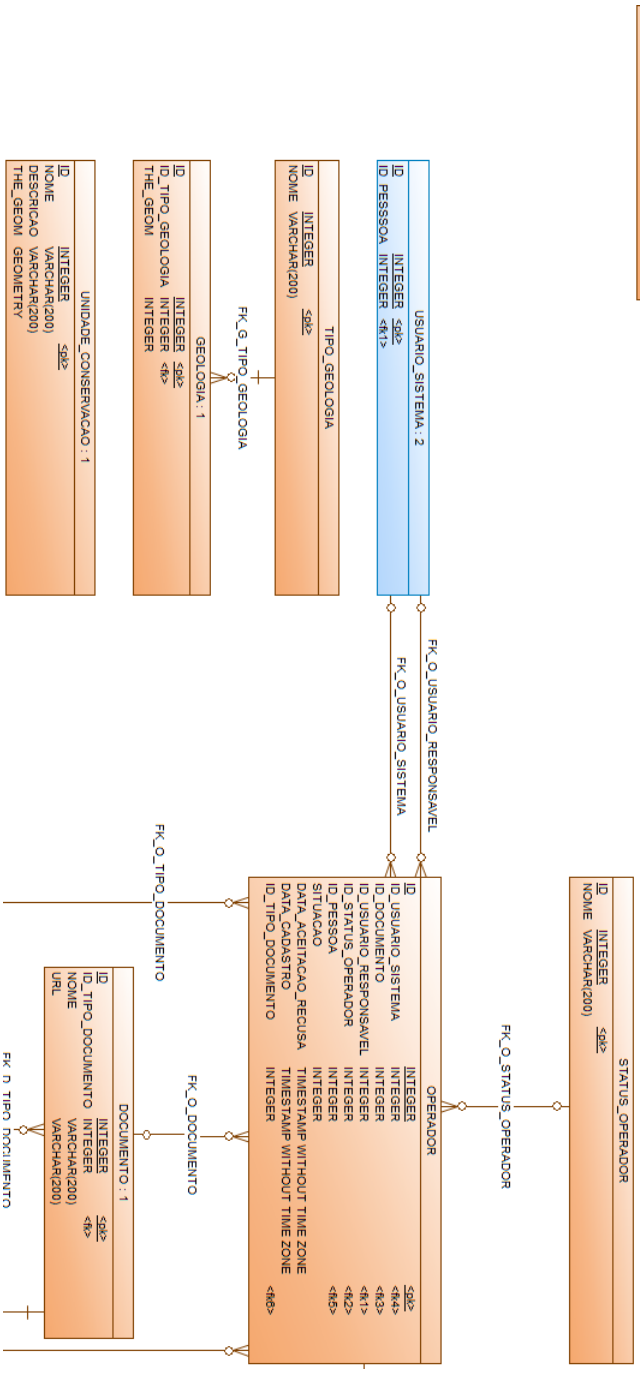
Quando cria-se um banco de dados, não parte-se para o *script* SQL de vez. Existe um processo indispensável para o bom funcionamento do banco de dados, que é o **planejamento**.

Um banco de dados bem projetado promove uma entrada e recuperação de dados consistente e reduz a existência de dados duplicados entre as tabelas do banco de dados. As tabelas de banco de dados relacionais trabalham juntas para garantir que os dados corretos estejam disponíveis quando necessário. Assim, pode-se por exemplo, planejar um banco de dados primeiro em papel.

É importante ouvir e discutir o propósito do banco de dados a ser desenvolvido, seja com o cliente ou com *Product Owner* (*P.O.*) como intermédio entre o DBA e o cliente. Isso se deve ao fato de que quando os detalhes são discutidos e entendidos de forma clara, as chances de ter uma nova discussão de que algo não está de acordo são menores. Assim, se estiver algo faltando, ou se pode melhorar algum item pedido, é importante apresentar ao cliente, afinal o cliente tem que estar satisfeito quando o sistema estiver pronto.

Outro item de importância é documentar o banco de dados. No LEMAF utiliza-se a modelagem conceitual de nível lógico do banco de dados. Mais especificamente utilizando-se a notação do **Diagrama de Engenharia da Informação** (ou ***Information Engineering Diagram*** - *IE Diagram*).

Figura 4.1 – Modelo de nível lógico usando o Diagrama IE.



Fonte: Do autor

De acordo com (MARTIN, 1989) o Diagrama IE pode ser descrito como um conjunto interligado de técnicas automatizadas em que modelos corporativos, modelos de dados e modelos de processos são construídos em uma base de conhecimento abrangente e são usados para criar e manter sistemas de processamento de dados. Na Figura 4.1 apresenta-se um dos tipos e modelos que utilizam o conceito do Diagrama IE que, neste caso, é voltado para a modelagem lógica de banco de dados. A Figura 4.1 está relacionada a um Diagrama de dados desenvolvido com base no SGBD PostgreSQL, onde no canto da imagem é denotado o esquema e abaixo as ligações entre algumas entidades.

O modelo lógico constitui uma representação específica de um modelo interno, utilizando as estruturas de banco de dados suportada pelo SGBD escolhido. Em um Banco de Dados Relacional, o esquema interno é expresso utilizando linguagem SQL, por padrão. Neste nível, o modelo lógico depende do *Software*. Portanto, qualquer alteração feita no SGBD exige que o modelo interno seja alterado para adequar-se às características e exigências de implementação referentes à modelagem do banco de dados. No entanto, o modelo lógico continua independente de hardware, ou seja, qualquer alteração (escolha de um computador, sistema operacional diferente, etc) não afetará o modelo lógico.

A partir do momento que o Diagrama IE está pronto, e que todas as pendências já foram resolvidas com o cliente, implementa-se o banco de dados e garante-se que este esteja disponível para os usuários. Em muitos Softwares de modelagem, é possível gerar a SQL a partir do Diagrama Lógico gerado, mas pela experiência, não é muito usual executar esse processo, pois gerar a SQL a partir do Diagrama IE pode trazer alguns objetos (tabelas, usuários, *constraints*, etc) ou uso da sintaxe que não é adequado.

Assim, é aconselhável criar o banco de dados manualmente pela escrita passo a passo da SQL. Um exemplo é não permitir que as definições de *index*, *tablespaces* e outros objetos assumindo nome e valores por *default*. O modo de como é criado o banco de dados é descrito posteriormente, onde é dado detalhes de qual é o processo seguidos pelos sistemas desenvolvidos pelo LEMAF.

Também é de responsabilidade do DBA definir os métodos de acesso aos bancos de dados. Este processo é realizado juntamente com os integrantes da Infraestrutura, pois eles sabem quais são os servidores adequados a serem utilizados pelos banco de dados. Logo, são eles quem informam o *host* de acesso para o servidor do banco de dados, além do usuário para conectar-se.

Por fim, documenta-se estes acessos, pois é necessário um documento definindo e explicando como acessar o banco de dados. É importante para a Empresa especificar tudo o que foi feito tanto para o banco de dados quanto para todo o sistema, porque deve-se entregar tudo o que foi desenvolvido, todo o processo realizado para atingir tal objetivo e como foi feito para o cliente, uma vez que ele é o usuário final.

Nesta atividade, o estagiário participou de todas as etapas junto à equipe de desenvolvimento e à equipe de infraestrutura.

4.4 Rotinas de *backups*

É importante desenvolver estratégia para recuperar os dados e manter a integridade dos mesmos caso uma falha catastrófica ocorra. Portanto, é necessário realizar uma **rotina de *backups***.

Segundo (DIGITAL, 2019), *backup* consiste em fazer uma ou mais cópias dos arquivos de dados a fim de preservar e recuperar esses dados em caso de perda por qualquer motivo. Há pontos que devem ser considerados no planejamento e execução de cópias de segurança. São eles:

> "**(i)** Se possível fazer mais de uma cópia, a redundância neste caso é bem vinda; **(ii)** Fazer *backup* e guardar o arquivo no mesmo computador ou servidor é uma prática perigosa, pois em caso de pane geral no HD ou roubo do *hardware*, por exemplo, você perde os dados originais e a cópia; **(iii)** Evite mídias pouco seguras como *pen drive* para fazer *backup*; **(iv)** Tenha um plano de *backup* que estabeleça como e quando as cópias deverão ser feitas. Uma ideia é fazer *backup* todos os dias e usar mídias diferentes para cada dia; **(v)** Mais importante do que ter um plano de *backup* é executá-lo fielmente" (DIGITAL, 2019).

Nos projetos que o estagiário trabalhou, apenas o SIOUT-RS possui rotina de *backup* que o LEMAF controla. Para os demais sistemas da equipe que pertencem ao estado do Pará, estes serviços foram terceirizados.

Código 4.1 – Código em Shell Script para realizar o *backup* da base de dados.

```
1 #!/bin/sh
2 #
3 # Script de backup do projeto
4
5 DATA=`date +%Y-%m-%d`
6 /usr/local/bin/pg_dump -U postgres -f /var/dados/
    backups/projeto_$DATA.backup
7 -F c -v "database" > /var/dados/backups/
    log_projeto_$DATA.log 2>&1
```

O processo realizado é feito a partir de um arquivo "*Shell Script*". **Shell Script** é uma linguagem interpretada e projetada para realizar uma

determinada tarefa utilizando comandos específicos do *bash* e os executáveis do Sistema Operacional. Com um arquivo executável, o usuário pode executar uma sequência de operações, instruções e testes. O Código 4.1 apresenta um código em *Shell Script* com um comando que tem por função gerar arquivos de *backup* para projetos que possuem o PostgreSQL como SGBD.

O comando `pg_dump`[1] é um utilizado para fazer *backup* lógico de um banco de dados do PostgreSQL. Ele faz *backups* consistentes, mesmo se o banco de dados estiver sendo usado simultaneamente. O `pg_dump` não bloqueia outros usuários que acessam o banco de dados (leitores ou gravadores). Este comando possui as suas opções para realizar o *backup*. Estas opções variam diante da necessidade do Administrador de banco de dados. São descritas aqui, de acordo com (GROUP, 2016), as opções que são usadas pelos colabores do LEMAF. São elas:

- `-U username`: nome de usuário para se conectar com o banco de dados;
- `-f file`: fornecido para especificar o nome do arquivo de *backup* juntamente com seu diretório;
- `-F format`: seleciona o formato da saída. Este comando deve ser seguido de uma das quatro opções que ele recebe. No Código 4.1 é utilizada a opção `c` que é a saída de um arquivo de formato customizado adequado para entrada no `pg_restore` (que é o comando para restaurar um *backup*). Juntamente com o formato de saída do diretório, esse é o formato de saída mais flexível, pois permite a seleção manual e a reordenação de itens arquivados durante a restauração;

[1] https://www.postgresql.org/docs/9.5/app-pgdump.html

- `-v`: especifica a descrição de saída, descrevendo todo o processo de *backup*. Isso fará com que o `pg_dump` produza comentários detalhados sobre o objeto e inicie ou interrompa o arquivo de despejo, e progrida as mensagens para o erro padrão; e
- `dbname`: especifica o nome do banco de dados no qual se quer extrair o *backup* dos dados.

Figura 4.2 – Parâmetros utilizados ao criar um Cron Job.

```
┌──────────── min (0 - 59)
│ ┌────────── hora (0 - 23)
│ │ ┌──────── dia do mês (1 - 31)
│ │ │ ┌────── mês (1 - 12)
│ │ │ │ ┌──── dia da semana (0 - 6) (0 a 6 representa
│ │ │ │ │                                de domingo a sábado, ou use nomes;
│ │ │ │ │                                 7 também representa domingo)
│ │ │ │ │
0 1 * * * /var/dados/backups/scripts/rotina_backup.sh
```

Como é necessário executar o *script* como uma rotina, precisa-se de um agendador de tarefas para programar a execução deste *script*. Para isso, utiliza-se o agendador de tarefas *Cron*. O *Cron* é usado para agendar comandos em um horário específico. Esses comandos ou tarefas agendados são conhecidos como *Cron Jobs* (G., 2018). O Cron geralmente é usado para executar *backups* planejados, monitorar o espaço em disco, excluir arquivos (por exemplo, arquivos de *log*) periodicamente que não são mais necessários, executar tarefas de manutenção do sistema e outros. A Figura 4.2 apresenta a criação de um *Cron Job* usando como primeiro parâmetro o dígito 0 que representa em qual minuto será executado. Logo após, o dígito 1 representa a hora de execução e, por fim, o dígito * (asterisco) representando que o *Cron Job* irá executar todos os dias dos mês, todos os meses e todos os dias da semana. Por fim, o próximo parâmetro contém o diretório do arquivo no qual será executado.

Nesta atividade, o estagiário participou parcialmente junto à equipe de infraestrutura do LEMAF.

4.5 Atualização de servidores de Desenvolvimento, Homologação e Produção

Na área de TI os projetos passam por um estágio até chegar a sua versão final e ser entregue ao cliente. Primeiro, há o acordo entre o cliente e a empresa para decidir o que é o projeto, qual o seu objetivo e o que se espera das funcionalidades. A partir disso, a equipe para qual foi designado o projeto irá começar a desenvolver até o prazo estipulado. Para que o projeto seja entregue com o que foi pedido e com um bom funcionamento, o mesmo precisa passar por testes até chegar em produção. Assim, existem os servidores de desenvolvimento, homologação e produção.

O ambiente de desenvolvimento é um ambiente de testes para a própria equipe testar o funcionamento de todas as novas funcionalidades antes de publicar a nova versão da aplicação para a utilização dos usuários finais. É importante ter um ambiente de desenvolvimento, pois é uma forma de evitar erros que causam impacto.

O ambiente de homologação é um ambiente para testes, porém este ambiente é usado tanto para a equipe quanto para o cliente testar as funcionalidades que já passaram por testes pela equipe, mas que devem ser aprovadas pelo cliente. Portanto, nesta fase o cliente irá aprovar ou não se a modificação realizada é favorável ao que foi solicitado. Se aprovado será publicado em produção, caso contrário voltará para equipe que terá que corrigir a inconsistência.

O ambiente de produção é o ambiente que disponibiliza a versão para o usuário final. Tudo que está presente na versão da aplicação do ambiente de produção é passado por aprovação dos clientes após passar pelo ambiente de homologação.

Assim como os desenvolvedores, os DBAs também precisam executar as *evolutions*, que é uma série de *scripts* SQL que contemplam as configurações do banco de dados, para manter a base de dados de acordo com as novas funcionalidades da aplicação. O processo de execução é manual. Logo, sempre que é necessário executar uma *evolution* em algum ambiente, alinha-se com os desenvolvedores qual ou quais devem ser executadas. Ainda está em fase de estudo para automatização a execução em algum ambiente.

Como a cada *release* do projeto pode-se ter um número grande ou não de *evolutions* a serem executadas, tem-se que agendar os procedimentos de execução, pois há a possibilidade de ocorrer algum imprevisto que impeça ou atrase esse procedimento. Os *scripts* podem gerar erro na execução por falta da validação feita de forma correta fazendo com que perca um tempo para corrigir e executar.

Outro ponto importante neste processo é que nem sempre há alguém disponível para executar os *scripts* no momento que é necessário. Pode ser que todos os desenvolvedores estejam executando outras atividades e que são forçados a parar o que estão desenvolvendo para realizar esse processo. Logo, sempre é passado para os desenvolvedores resolver todos os impedimentos em outras atividades, para quando chegar o momento de executar as *evolutions* estejam preparados apenas para essa atividade.

É necessário manter o controle das *evolutions* criadas e a execução das mesmas em outro ambiente. Para manter este controle são criadas pla-

Figura 4.3 – Planilha de controle de *evolutions*.

EVOLUTION	BRANCH	Servidor de Desenvolvimento	Servidore de Homologação	Servidor de Produção
1	master	X	X	X
2	master	X		
3	master	X	X	X
4	master	X	X	
5	master	X	X	X
6	master	X	X	X

Fonte: Do autor

Figura 4.4 – Planilha de controle de *evolutions*.

COMENTÁRIO GIT	DATA	SOLICITADO POR	EXECUTADO POR	COMENTARIO ESTRUTURA	MODELO	DIC. DADOS	OBSERVAÇÕES
Comentario da primeira evolution	24/09/2018	João	Igor Nunes	X			https://taiga.ti.lemaf.ufla.br/project/magaleo-manutencao-simlam-sisflora/task/1082
Comentário da segunda evolution	08/10/2018	Carlos	Igor Nunes	X			Validado por Lucas
Comentário da terceira evolution	22/10/2018	Otávio	Igor Nunes	X			Validado por Matheus
Comentário da quarta evolution	01/11/2018	Gabriel	Igor Nunes	X			https://taiga.ti.lemaf.ufla.br/project/magaleo-manutencao-simlam-sisflora/task/1225
Comentário da quinta evolution	13/11/2018	Rodrigo	Igor Nunes				Validado por Italo
Comentário da sexta evolution	13/11/2018	Paulo	Igor Nunes	X			Validado por Saulo

Fonte: Do autor

nilhas de controle para armazenar número máximo de *evolutions* e não criar uma nova com a mesma numeração de alguma já existente, como apresentado na Figura 4.3 e Figura 4.4.

Nas planilhas, é informado a *branch* do **GitLab**[2] onde ela foi criada e colunas contendo todos os servidores dos ambientes das base de dados onde estão marcados com um "X" para indicar em qual servidor os *scripts* já foram executados. Além disso, há uma coluna que indica o comentário que foi utilizado para enviar o arquivo ou arquivos para o repositório no GitLab. Logo após, possui a coluna que indica a data em que ocorreu a criação da *evolution*, seguido do nome de quem solicitou e quem executou

[2] https://about.gitlab.com/

a mesma em algum ambiente. Além dessas colunas, há outras três que indicam se houve comentários em esquemas e entidades que foram criadas, se os mesmos já estão disponíveis no dicionário de dados gerado e se já estão disponíveis em Diagrama IE. É necessário manter o dicionário e o Diagrama IE atualizados para, quando solicitado, estiverem com a última versão da base de dados. Por fim, a última coluna das planilhas é um campo para observações. Nesta coluna coloca-se o nome de quem validou esta *evolution* ou um endereço Web do Taiga referente a tal atividade.

Nesta atividade, o estagiário participou de todas as etapas para a sua realização.

4.6 Ligação entre Teoria e Prática

Ingressando em um estágio, o estagiário deve fazer bom uso do que foi aprendido nas disciplinas durante a graduação. Assim, faz-se uma ligação entre a teoria (disciplinas) e a prática (estágio).

Sabe-se que para uma vaga em uma empresa é necessário escolher a área na qual tem-se mais aptidão para que o estagiário cresça tanto profissionalmente quanto pessoalmente. Logo, nem todo o conhecimento absorvido de todas as disciplinas da graduação terão uma conexão durante o período de estágio.

Para o estagiário, quatro disciplinas foram cruciais para que estivesse habilitado a atuar na área de banco de dados no LEMAF. Banco de Dados I e Banco de Dados II deram a base de conhecimentos tanto para criação de *scripts* SQL quanto para ter noção de leitura e escrita do SGBD no disco, modelagem do banco de dados, otimização de consultas, etc. A disciplina de Engenharia de *Software* auxiliou para o conhecimento sobre

como um *Software* é criado, seus casos de uso, diagrama de classes, métricas, entre outros. E por fim, a disciplina de Projeto de Gerência de *Software* deu uma visão sobre o ciclo de vida de um projeto, quais a fases em que o projeto passa antes de chegar ao cliente, como as atividades são distribuídas com o auxílio de metodologias ágeis, entre outros.

Logo, o estágio abre portas para que o aluno de graduação possa aplicar o conhecimento adquirido em diversas disciplinas, pois é necessário ver como a rotina no mercado de trabalho flui em volta de toda teoria aprendida.

5 IMPLEMENTAÇÃO DE EVOLUTIONS PARA A MANUTENÇÃO DE BANCO DE DADOS DE SISTEMAS AMBIENTAIS

O objetivo do banco de dados é, dentre outros, o de manter a consistência e a segurança dos dados para o bom funcionamento dos sistemas os quais pertence. Tendo isso em vista, tem-se o conceito de *Evolutions*. As *evolutions* contemplam uma série de *scripts* SQL que armazenam as modificações de esquema, executadas sobre a base de dados. Estas modificações são importantes para o projeto, pois a partir do momento que o sistema for migrado para outro ambiente (ambiente de produção, por exemplo), a base de dados deve manter a mesma estrutura. As *evolutions* são criadas durante o desenvolvimento do sistema. Logo, não há *evolution* desenvolvida sem que o sistema precise.

O termo *Evolution* é referente a um *framework* denominado *Play*. O *Play* é uma estrutura de aplicativos de alta produtividade que integra componentes e APIs para o desenvolvimento moderno de aplicativos da *WEB* (PLAY, 2019).

Com o auxílio do *Play* é possível realizar a sincronização das *evolutions* com as configurações do banco de dados entre os ambientes de desenvolvimentos por meio da execução automatizada dos *scripts*.

Código 5.1 – Exemplo de *evolution* para o banco de dados para os sistemas do LEMAF.

```
1  # --- !Ups
2
3  CREATE ROLE role1 LOGIN
4    ENCRYPTED PASSWORD 'password'
5    NOSUPERUSER INHERIT NOCREATEDB NOCREATEROLE NOREPLICATION;
6
7  CREATE SCHEMA schema1;
8  ALTER SCHEMA schema1 OWNER TO owner_schema;
9
10 # --- !Downs
11
```

```
12  DROP ROLE IF EXISTS role1;
13
14  DROP SCHEMA IF EXISTS schema1;
```

A primeira *evolution* criada para o projeto contém apenas criações de usuários (*roles*) que são necessárias para a comunicação entre a aplicação e o banco de dados. Esta *evolution* contém as devidas cláusulas indicando o que o usuário pode fazer ou não e, a criação dos esquemas (Código 5.1), usando o PostgreSQL como exemplo. As demais *evolutions* criadas a partir da primeira são para criação de tabelas, *stored procedures*, *functions*, *views*, etc.

No Código 5.1 é apresentada uma estrutura que separa o *script* em duas partes: o ***!Ups*** e o ***!Downs***. O *!Ups* deve conter sempre o que vai evoluir (ou seja, alterar) a base de dados, tanto para adicionar quanto para remover algum tipo de dado ou estrutura. O *!Downs* é exatamente o contrário (desfazer a alteração). Assim, o *script* SQL reverte tudo o que foi realizado no *!Ups*.

!Ups e *!Downs* são diretivas utilizadas pelo *Play* como um padrão para identificar qual o *scritp* que irá realizar a modificação na base dados e qual irá desfazer tal modificação. No Código 5.1 pode-se notar que o *!Ups* e *!Downs* se encontram em um mesmo arquivo. Pode ser um problema para execução automatizada *evolutions* mas este *framework* consegue identificar pelas diretrizes o que deve ser executado. Logo, não haverá a execução do *!Ups* e *!Downs*, apenas do *!Ups*.

Um dos motivos para utilizar este tipo de estrutura é o de manter a integridade dos dados, pois se por algum motivo o *!Ups* da *evolution* causar alguma inconsistência na base de dados, há como executar o *!Downs* desfazendo esta alteração e voltando para seu estado imediatamente anterior.

O Código 5.1 mostra o comando `CREATE ROLE` que tem como objetivo criar um usuário para que este tenha acesso à base de dados. No LEMAF, geralmente este comando é seguido das seguintes cláusulas (no momento da criação): `NOSUPERUSER`, `INHERIT`, `NOCREATEDB`, `NOCREATEROLE` e `NOREPLICATION`. A cláusula `NOSUPERUSER` tem como objetivo não deixar que este usuário tenha permissão de superusuário sobre a base de dados, ou seja, não ter privilégios para administração de banco de dados. Geralmente tem-se apenas um superusuário para não causar inconsistências na base de dados. A cláusula `INHERIT` que tem como objetivo "herdar" privilégios sobre objetos do banco de dados de usuários na qual é diretamente ou indiretamente membro. A cláusula `NOCREATEDB` tem como objetivo não permitir que o usuário crie uma base de dados. O mesmo ocorre com a cláusula `NOCREATEROLE` na qual não permite que o usuário crie outro usuário. Por fim, a cláusula `NOREPLICATION` não permite que o usuário faça replicação de dados da base de dados. Nos projetos do LEMAF é comum que apenas o superusuário tenha os privilégios de replicação e criação de novos bancos de dados.

Em seguida, o Código 5.1 mostra o *script* SQL de como é criada uma *evolution*. No *!Ups*, o primeiro a se fazer é criar os esquemas. Para isso, usa-se `CREATE SCHEMA`. Um esquema é semelhante a uma classe tornando-se, um ponto positivo, pois deixa a base de dados estruturada contendo todas as entidades em comum em cada esquema. Um esquema pode conter uma ou mais entidades. Logo em seguida, o comando `ALTER SCHEMA` tem como objetivo alterar o usuário dono deste esquema, ou seja, este usuário terá permissões superiores aos demais. Este comando é seguido do comando `OWNER TO`, que indica o novo dono do esquema.

No *!Downs* desfaz-se todas as modificações na base dados realizadas no *!Ups*. Logo, o comando `DROP SCHEMA` tem como objetivo remover o esquema da base de dados. O mesmo ocorre com o comando `DROP ROLE` que tem como objetivo remover o usuário da base de dados.

As *evolutions* iniciais criadas em relação a criação de esquemas e tabelas são aquelas que são a base para as primeiras versões do sistema. Logo, são do modelo de dados criado pelos DBAs juntamente com os desenvolvedores. Após as *evolutions* essenciais para o funcionamento do sistema, as demais *evolutions* são executadas sob demanda dos desenvolvedores ou dos clientes.

Código 5.2 – Padrões de evolutions para os desenvolvedores

```
1  # --- !Ups
2
3  CREATE TABLE  esquema.tabela(
4
5  id BIGSERIAL/INTEGER NOT NULL,
6  id_tabela_alvo INTEGER,
7  id_tabela_outro_esquema INTEGER NOT NULL,
8  CONSTRAINT pk_tabela PRIMARY KEY(id),
9  CONSTRAINT fk_t_tabela_alvo FOREIGN KEY(id_tabela_alvo)
10  REFERENCES esquema.tabela_alvo(id)
11
12 );
13
14 GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE  esquema.tabela TO usuario;
15 GRANT SELECT, UPDATE ON SEQUENCE  esquema.tabela_id_seq TO usuario;
16
17 COMMENT ON TABLE esquema.tabela IS 'Entidade responsavel por ...';
18 COMMENT ON COLUMN esquema.tabela.id IS 'Identificador unico da entidade.';
19 COMMENT ON COLUMN esquema.tabela.id_tabela_alvo IS 'Comentario do campo. Identificador da entidade
       esquema.tabela_alvo';
20 COMMENT ON COLUMN esquema.tabela.id_tabela_outro_esquema  IS 'Comentario do campo.Identificador da
       entidade esquema2.tabela_outro_esquema';
21
22 # --- !Downs
23
24 DROP TABLE esquema.tabela;
```

Código 5.3 – Criando sequência em script SQL

```
1  CREATE SEQUENCE esquema.sq_nome_da_tabela
2    MINVALUE 1
```

```
3     MAXVALUE 9223372036854775807
4     START 1
5     CACHE 1;
6
7   CREATE TABLE esquema.nome_da_tabela (
8       id BIGINT NOT NULL DEFAULT nextval('esquema.sq_nome_da_tabela'::regclass),
9       ...
10  );
```

No LEMAF os DBAs nem sempre desenvolvem as *evolutions*, mas sempre as validam. Em muitas ocasiões, os desenvolvedores realizam os *scripts* das *evolutions* e enviam para avaliar se foi feito de maneira correta. Para não ficar muito tempo corrigindo os *scripts*, escreve-se um documento, acessível para todos os desenvolvedores, com um padrão de escrita e tipos dos atributos das entidades para a validação ser feita de maneira mais rápida e dar mais vazão entre as atividades. Este, também é fornecido para novos DBAs para aprenderem como é o padrão de desenvolvimento dos *scripts*. O *script* apresentado no Código 5.2 representa o padrão utilizado para as *evolutions* que consta no documento de padrões. Sempre que uma entidade é criada, deve-se descrever o seu nome com `SCHEMA.tabela` significando a qual esquema a entidade pertence.

O *id* das entidades (a *PRIMARY KEY*) pode variar entre *idt* seguido pelo nome da entidade ou apenas *id*. Isso pode variar de acordo com o padrão de cada equipe, mas não é outro tipo de nome. Além disso, o tipo de dado deste atributo pode ser *INTEGER* caso tenha-se certeza de que a quantidade de tuplas da entidade é um número fixo. Para a indicação de um valor sequencial, pode-se utilizar o tipo de dado *SERIAL* que é um pseudo-tipo para definir colunas auto incrementáveis, permitindo que a cada inserção na tabela a coluna possua um valor único. Também possível criar colunas auto incrementáveis utilizando *SEQUENCE*[1]. Este, atua da mesma forma que

[1] https://www.postgresql.org/docs/9.5/sql-createsequence.html

o pseudo-tipo citado anteriormente, mas a *SEQUENCE* precisa ser criada antes de ser atribuída a um identificador da tabela, sendo necessário atribuir valores iniciais aos parâmetros da sequência.

O Código 5.3 contém um exemplo da criação de uma *SEQUENCE* onde pode-se notar que alguns dos parâmetros citados anteriormente foram utilizados. Algumas equipes da qual o estagiário fez parte utilizam a sequência desta forma atribuindo como valor *default* para o atributo *id* da entidade criada na linha 8, atribuindo a este atributo com uma função chamada *nextval*. Essa função faz com que o próximo valor do atributo seja o valor atual da sequência em questão mas, para isso ser aceito pelo PostgreSQL precisa-se utilizar um *casting* para a sequência atribuída, pois a função aceita apenas parâmetro do tipo *regclass*. *Regclass* é um alias para *oid*, ou "identificador de objeto". O *Casting* para *regclass* é um atalho para informar "este é o nome de uma relação, por favor converta-a para o *oid* dessa relação". Abaixo é explicado cada parâmetro utilizado na criação de uma sequência utilizada na base de dados dos projetos que o estagiário atuou:

- *minvalue* ou *NO MINVALUE*: Esta cláusula é opcional. Indica se a sequência terá um valor mínimo (*minvalue*) ou não (*NO MINVALUE*);

- *maxvalue* ou *NO MAXVALUE*: É uma cláusula opcional. Indica se a sequência terá um valor máximo (*maxvalue*) ou não (*NO MAXVALUE*)

- *start*: A cláusula opcional *START WITH start* permite que a sequência inicie por qualquer valor. Por padrão, o valor de início para

sequências crescentes é *minvalue* enquanto *maxvalue* é o valor padrão para sequências decrescentes;

- *cache*: A cláusula opcional *CACHE cache* especifica quantos números da sequência serão pré-alocados e armazenados na memória. O valor mínimo e padrão é 1;

Continuando a análise do Código 5.2, para as chaves estrangeiras e as chaves primárias de outras entidades que estão sendo referenciadas, segue-se um padrão para o seu nome. Tanto para entidades do mesmo esquema quanto para entidades de outro esquema, costuma-se nomear estas chaves com nomes diferentes. Para entidades que pertencem ao mesmo esquema, estas são nomeadas como *id* seguido com o nome da entidade a ser referenciada. Já para entidades que pertencem a outro esquema, nomeia-se com *id* seguido do nome do esquema para tornar visualmente fácil a diferença de entidades de mesmo esquema ou não. Como os desenvolvedores precisam mapear estas chaves estrangeiras no código da aplicação, também deve-se preocupar com o nomes das restrições de integridade referencial usando a cláusula *Constraint*. Portanto, seus nomes também possui um padrão que tem como início *pk* seguido do nome da tabela para chaves primárias. Para chaves estrangeiras o nome usado é *fk* seguido da inicial ou das iniciais da entidade na qual está referenciando seguida do nome da entidade referenciada.

Outro ponto a ser citado e não menos importante, sempre que cria-se novas entidades e seus atributos, utiliza-se os comentários para esclarecer o que a entidade tem como objetivo e descrever o que cada atributo significa. Geralmente, empresas têm uma rotatividade de colaboradores e isso auxilia um novo colaborador a entender cada entidade e seus atributos.

Por fim, concede-se os privilégios para cada usuário que utiliza as informações das entidades e esquemas da base de dados. Os DBAs sempre criam uma nova *evolution* utilizando o superusuário. Como o superusuário tem o privilégio de *owner* de todos os esquemas e entidades, não é necessário conceder nenhum privilégio, pois ele já os possui. Já os demais usuários precisam destes privilégios. Porém, não são todos, apenas o suficiente para que os mesmos possam operar. Logo, sempre que há uma nova entidade ou esquema, concede-se os privilégios de *INSERT*, *DELETE*, *UPDATE*, *SELECT* e *USAGE* para os usuários da aplicação e apenas *SELECT* e *USAGE* para usuários externos, ou seja, não nativos da aplicação. Este caso citado é usado apenas para o SGBD PostgreSQL. Para o Oracle, utiliza-se o mesmo usuário da aplicação. O superusuário do Oracle, para os sistemas do LEMAF, é apenas para administração e não tem acesso às entidades que são utilizadas para a aplicação.

6 CONSIDERAÇÕES FINAIS

O período de estágio realizado na empresa LEMAF foi um momento de grande aprendizado e experiência para o desenvolvimento profissional do estagiário. Isso é importante principalmente para alguém que está seguindo a carreira no mercado de trabalho, pois irá vivenciar momentos no qual já passou, além de desafios que talvez saberá solucionar ou traçar um caminho para que ele seja resolvido. Isso foi e será importante para atingir um degrau a mais em sua base de conhecimento, tornando-o um profissional mais capacitado.

Neste período, foi importante enxergar que os conceitos aplicados ao aluno em sala de aula favoreceram ao estagiário aplicá-los em seu ambiente de trabalho, permitindo que o aprendizado na graduação pudesse ser colocados em prática. É muito gratificante ver que os ensinamentos dos professores valeram muito a pena.

Vale ressaltar que durante o início do estágio, dificuldades foram encontradas em alguns momentos por não conhecer a fundo ou ter nenhum conhecimento sobre algumas tecnologias e ferramentas utilizadas para desenvolvimento dentro da empresa. PostgreSQL e Oracle eram novidades naquele momento por ter apenas trabalhado com MySQL. Porém, com o tempo, as dificuldades estavam se tornando conhecimento e mesmo sabendo que SGBDs são muito semelhantes principalmente em seus conceitos, há algumas diferenças. Também não ter vivenciado o ciclo de vida de um projeto por meio das metodologias ágeis e conhecer termos utilizados, estimativas, retrospectivas, entre outras foi um momento de dificuldade para entender como é o funcionamento deste processo na prática, mas abriu

os olhos para ver o quão necessário é utilizar esse processo para o desenvolvimento de projetos que vão além da área de Tecnologia da Informação.

Destaca-se alguns pontos observados e que podem ser melhorados, a fim de otimizar os processos internos da empresa:

- a falta de servidores e falta de espaço nos servidores que já possui para armazenamento de arquivos importantes, como *backups*, arquivos de informações geométricas (mapas), são importantes, para manter a segurança dos dados e, portanto, deixar de serem armazenados em máquinas locais;

- diagramas de bancos de dados que são um dos documentos importantes para as entregas de projetos ainda são feito de forma manual. Logo, encontrar algum *software*, de preferência *open source*, para gerá-los automaticamente é de grande ajuda; e

- *evolutions* ainda são executadas de forma manual. Logo, encontrar alguma estratégia para a execução automatizada dos *scripts*.

Tendo em mente que são importantes, a equipe de banco de dados apresentou estes pontos apresentados aos seus respectivos Gerentes de Projetos. O Gerente de Projetos da equipe do estagiário analisou todos os pontos que podem ser melhorados e os enviou à Coordenadoria para análise da viabilidade.

Quanto às melhorias propostas para o bem-estar não só da equipe como da empresa, são ressaltados os seguintes pontos:

- é necessário analisar alguma forma que não cause tanto impacto para atualizar as versões dos SGBDs, pois alguns estão em versões antigas

e perdendo as novas funcionalidades de versões mais recentes que podem ajudar no desempenho da aplicação;

- é importante implantar um sistema ou fazer uso de algum *framework*, por exemplo, o *Play*, para automatização da execução de *evolutions* em ambientes de testes, homologação e produção, pois executar de forma manual custa tempo e há a possibilidade de esquecer de executar alguma;

- incluir na ferramenta de geração de dicionário de dados a geração de dicionário de dados dos SGBDs Oracle, uma vez que a ferramenta gera apenas para bases de dados do PostgreSQL. Realizar esse processo manualmente gera um custo muito alto de tempo;

- procurar sempre utilizar o tipo de dados *SERIAL* em vez de criar *sequences* e atribuir para as chaves primárias de entidades que possuem auto incremento, para SGBDs PostgreSQL, com o uso padronizado do tipo *SERIAL* há menos garantias de retornar erros; e

- é necessário uma ferramenta para implementar os Diagramas de Dados das bases de dados do sistemas. Não se tem uma ferramenta padrão para modelagem na qual a equipe de banco de dados faz uso. Para PostgreSQL pode-se propor duas ferramentas: o pgModeler[1] que é uma ferramenta *Open Source* e gratuita e o DataGrip[2] que gera o diagrama a partir do esquema selecionado, mas deve-se ter o banco de dados já implementado, além de ser uma ferramenta paga. E para

[1] https://pgmodeler.io/
[2] https://www.jetbrains.com/datagrip/

Oracle sugere-se o SQL Developer Data Modeler[3] que é um produto Oracle, apesar de ser uma ferramenta paga.

As melhorias propostas são de pequenos pontos que podem impulsionar ainda mais a empresa dentro do mercado inserido e auxiliá-la a criar um melhor ambiente de trabalho, visando o compartilhamento de conhecimentos e trocas de experiências.

Por fim, vale ressaltar a importância do estágio para o aprimoramento acadêmico e profissional de qualquer aluno, já que através dele aprimora-se capacidades técnicas, pessoais e sociais, tornando o aluno preparado, sabendo que terá de resolver os problemas que virão, lidar com a pressão, saber ouvir opiniões diferentes que possam ajudar, entre outros diferenciais que o aluno terá para o mercado de trabalho.

[3] https://www.oracle.com/br/database/technologies/appdev/datamodeler.html

REFERÊNCIAS

ALFRESCO. **Overview**. 2019. (Acessado em 14/05/2019). Disponível em: <https://docs.alfresco.com/4.0/concepts/system-about.html>.

AMBIENTAL, A. **Adequação Ambiental**. 2019. (Acessado em 12/06/2019). Disponível em: <http://monitoramento.semas.pa.gov.br/AdequacaoAmbiental/#/login>.

ASCOM. **Sistema de gestão de recursos hídricos é a nova ferramenta de controle ambiental do Pará**. 2017. (Acessado em 03/10/2019). Disponível em: <https://www.semas.pa.gov.br/2017/12/31/sistema-de-gestao-de-recursos-hidricos-e-a-nova-ferramenta-de-controle-ambiental-do-para/>.

CARVALHO, V. **PostgreSQL: Banco de Dados para Aplicações Web Modernas**. [S.l.]: Casa Do Código, 2017.

DIGITAL, P. **Importância do banco de dados e do backup**. 2019. (Acessado em 27/03/2019). Disponível em: <https://www.luis.blog.br/importancia-do-banco-de-dados-e-do-backup/>.

ESPINHA, R. G. **Kanban: o que é e TUDO sobre como gerenciar fluxos de trabalho**. 2019. (Acessado em 28/05/2019). Disponível em: <https://artia.com/kanban/>.

G., A. **Como Configurar Cron Jobs no Linux (VPS)**. 2018. (Acessado em 12/05/2019). Disponível em: <https://www.hostinger.com.br/tutoriais/como-configurar-cron-jobs-no-linux/>.

GONÇALVES, E. **SQL: Uma Abordagem Para Banco de Dados Oracle**. SP, Brazil: Casa Do Código, 2014.

GROUP, T. P. G. D. **pg_dump - PostgreSQL 9.5.20 Documentation**. 2016. (Acessado em 15/04/2019). Disponível em: <https://www.postgresql.org/docs/9.5/app-pgdump.html>.

MARTIN, J. **Information Engineering - Book 1**. [S.l.]: Prentice Hall, 1989.

PLAY. **Play - Documentation**. 2019. (Acessado em 12/04/2019). Disponível em: <https://www.playframework.com/documentation/2.7.x/Introduction>.

PÚBLICO, S. **SIMLAM Público**. 2019. (Acessado em 12/06/2019). Disponível em: <https://monitoramento.semas.pa.gov.br/simlam/index.htm>.

REDAÇÃO. **Servidor Linux ou Windows? Entenda as diferenças!** 2019. (Acessado em 11/03/2019). Disponível em: <https://www.melhorhospedagemdesites.com/hospedagem-de-sites/servidor-linux-ou-windows/>.

ROUSE, M. **GitLab**. 2018. (Acessado em 27/05/2019). Disponível em: <https://whatis.techtarget.com/definition/GitLab>.

SCRUM. 2019. (Acessado em 29/05/2019). Disponível em: <https://www.desenvolvimentoagil.com.br/scrum/>.

SIGERH-PA. **SIGERH-PA**. 2019. (Acessado em 12/06/2019). Disponível em: <http://sistemas.semas.pa.gov.br/sigerhpa/>.

SIOUT-RS. **SIOUT-RS**. 2019. (Acessado em 12/06/2019). Disponível em: <http://www.siout.rs.gov.br>.

SISFLORA. **SISFLORA**. 2019. (Acessado em 12/06/2019). Disponível em: <https://monitoramento.semas.pa.gov.br/sisflora/>.

APÊNDICE A – Código SQL - Primeira *Evolution*

```
1  # --- !Ups
2
3  CREATE ROLE role1 LOGIN
4     ENCRYPTED PASSWORD 'password'
5     NOSUPERUSER INHERIT NOCREATEDB NOCREATEROLE
          NOREPLICATION;
6
7  CREATE SCHEMA schema1;
8  ALTER SCHEMA schema1 OWNER TO owner_schema;
9
10 CREATE SCHEMA schema2;
11 ALTER SCHEMA schema2 OWNER TO owner_schema;
12
13 CREATE SCHEMA schema3;
14 ALTER SCHEMA schema3 OWNER TO owner_schema;
15
16 CREATE SCHEMA schema4;
17 ALTER SCHEMA schema4 OWNER TO owner_schema;
18
19 CREATE SCHEMA schema5;
20 ALTER SCHEMA schema5 OWNER TO owner_schema;
21
22 # --- !Downs
23
24 DROP ROLE IF EXISTS role1;
25
26 DROP SCHEMA IF EXISTS schema1;
```

```
27
28 DROP SCHEMA IF EXISTS schema2;
29
30 DROP SCHEMA IF EXISTS schema3;
31
32 DROP SCHEMA IF EXISTS schema4;
33
34 DROP SCHEMA IF EXISTS schema5;
```

APÊNDICE B – Código SQL - Padrão de *Evolutions*

```
1  -- PADRAO DE CRIACAO DE SCRIPTS (EVOLUTIONS) PARA O
       BANCO DE DADOS:
2
3  # --- !Ups
4
5  CREATE TABLE esquema.tabela(
6
7  id BIGSERIAL/INTEGER NOT NULL, -- PRIMARY KEY
8  id_tabela_alvo INTEGER, -- FOREIGN KEY DO MESMO
       ESQUEMA
9  id_tabela_outro_esquema INTEGER NOT NULL, -- FOREIGN
       KEY DE OUTRO ESQUEMA
10 CONSTRAINT pk_tabela PRIMARY KEY(id), -- CONSTRAINT
       PRIMARY KEY
11 CONSTRAINT fk_t_tabela_alvo FOREIGN KEY(id_tabela_alvo
       ) -- CONSTRAINT FOREIGN KEY
12  REFERENCES esquema.tabela_alvo(id)
13
14 );
15
16 GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE esquema
       .tabela TO usuario; -- GRANT PARA O USUARIO NA
       TABELA
17 GRANT SELECT, UPDATE ON SEQUENCE esquema.
       tabela_id_seq TO usuario; -- GRANT PARA O USUARIO
       NA SEQUENCIA DA TABELA
18
```

```
19 COMMENT ON TABLE esquema.tabela IS 'Entidade
       responsavel por ...';
20 COMMENT ON COLUMN esquema.tabela.id IS 'Identificador
       unico da entidade.';
21 COMMENT ON COLUMN esquema.tabela.id_tabela_alvo IS '
       Comentario do campo. Identificador da entidade
       esquema.tabela_alvo';
22 COMMENT ON COLUMN esquema.tabela.
       id_tabela_outro_esquema  IS 'Comentario do campo.
       Identificador da entidade esquema2.
       tabela_outro_esquema';
23
24 UPDATE esquema.tabela_ja_existe SET id_tabela=%VALOR%
       [WHERE campo=%PARAMETRO DE UPDATE%];
25
26 ALTER TABLE esquema.tabela ALTER COLUMN id_tabela SET
       NOT NULL;
27
28 ALTER TABLE esquema.tabela_ja_existe ADD COLUMN
       id_tabela INTEGER;
29 ALTER TABLE esquema.tabela_ja_existe ADD CONSTRAINT
       fk_tje_tabela FOREIGN KEY(id_tabela)
30 REFERENCES esquema.tabela(id);
31
32 # --- !Downs
33
34 ALTER TABLE esquema.tabela_ja_existe DROP COLUMN
       id_tabela;
```

```
35 DROP TABLE esquema.tabela;
```

APÊNDICE C – Código SQL - Criação de *Sequence*

```
1  CREATE SEQUENCE esquema.sq_nome_da_tabela
2    MINVALUE 1
3    MAXVALUE 9223372036854775807
4    START 1
5    CACHE 1;
6
7  CREATE TABLE esquema.nome_da_tabela (
8        id BIGINT NOT NULL DEFAULT nextval('esquema.
             sq_nome_da_tabela'::regclass),
9        ...
10 );
```