Revista Interdisciplinar do Pensamento Científico. ISSN: 2446-6778
Nº 1, volume 1, artigo nº 07, Janeiro/Junho 2015
D.O.I: http://dx.doi.org/10.20951/2446-6778/v1n1a7

UMA PROPOSTA DE AUDITORIA EM SISTEMAS JAVA COM HIBERNATE E ASPECTJ EM BANCO DE DADOS COM SUPORTE A SQL

Emanuel Bonfante Muniz[1]

Tecnologo em Análise e Desenvolvimento de Sistemas

Renato Sousa Botacim[2]

Bacharel em Sistemas de Informação

Bruno Missi Xavier[3]

Mestre em Pesquisa Operacional e Inteligência Computacional

Marcelo Schuster Albuquerque[4]

Mestre em Pesquisa Operacional e Inteligência Computacionalo

Resumo

O grande volume de informação gerada pelos setores operacionais das organizações e manutenidos pelos sistemas de informação gerenciais necessitam de um controle transacional mais apurado a fim de identificar movimentações equivocadas ou até mesmo fraudulentas. Este artigo tem o objetivo de descrever um *framework* de auditoria capaz de ser incorporado a uma aplicação com a vantagem do baixo acoplamento de código. O estudo de caso proposto neste trabalho descreve a utilização deste *framework* de auditoria em um sistema de informação escolar com grande nível de satisfação na recuperação das

[1]Centro Universitário São Camilo-ES, Análise e Desenvolvimento de Sistemas, Cachoeiro de Itapemirim, ES, emanuel.ebm.ti@gmail.com
[2]Centro Universitário São Camilo-ES, Análise e Desenvolvimento de Sistemas, Cachoeiro de Itapemirim, ES, botacim.renato@gmail.com
[3]Centro Universitário São Camilo-ES, Análise e Desenvolvimento de Sistemas, Cachoeiro de Itapemirim, ES, bmissix@gmail.com
[4]Centro Universitário São Camilo-ES, Análise e Desenvolvimento de Sistemas, Cachoeiro de Itapemirim, ES, marceloaschuster@gmail.com

transações ocorridas no sistema. A discussão sobre o controle e auditoria de transações em sistemas de informação colabora para o aumento da segurança e confiabilidade dos dados gerados nas organizações, tema este imprescindível na tomada de decisão a nível gerencial.

Palavras-chave: Auditoria; Java; Oracle; Hibernate; AspectJ.

Abstract

The sheer volume of information generated by the operating sectors of organizations and manutenidos by management information systems need a more accurate transactional control to identify faulty or even fraudulent transactions. This article aims to describe a framework for audit can be incorporated into an application with the advantage of loosely coupled code. The case study proposed in this paper describes the use of this framework in a system audit of school information with great satisfaction in the recovery of transactions that occurred in the system. The discussion about the control and auditing of transactions in information systems contributes to increased security and reliability of the data generated in organizations, this essential theme in the managerial decision-making level.

Keywords: Audit; Java; Oracle; Hibernate; AspectJ.

1. INTRODUÇÃO

A utilização da tecnologia da informação tem crescido e alcançado uma importância vital dentro das empresas. Branski, (2004) cita existir “cerca de 2,5 bilhões de documentos textuais disponíveis na internet com uma taxa de crescimento de 7,5 milhões ao dia”. Segundo Royal Pingdom, (2012) citado por Xavier; Silva; Gomes, (2013, p.85) “300 milhões de novos sites foram publicados na web no ano de 2011, totalizando 555 milhões de sites disponíveis para consulta”.

Neste contexto, é crescente o número de informações armazenadas em repositórios de dados, bases de dados ou banco de dados, sendo representado por uma coleção de dados relacionados que possuem um significado explícito e representam fatos do mundo real ou virtual (ELMASRI; NAVATHE, 2002). Um Sistema Gerenciador de Banco de Dados (SGBD) mantêm as informações armazenadas de maneira organizada, estruturada e relacionadas entre si, e sendo cada vez mais utilizadas nas mais diversas áreas, onde a integridade e confiabilidade dos dados precisam ser mantidas.

Para garantir a confidencialidade e integridade, faz-se necessários algumas medidas de controle e segurança dos sistemas em operação. E ficando a cargo da auditoria de sistemas validar e avaliar os resultados gerados, a segurança, a eficácia dos processos concluídos e a confiabilidade das informações (GIRON, 2003).

A auditoria de sistemas consiste em recolher e avaliar evidências para garantir que a integridade dos dados seja mantida e, se houver adulteração, apontar com exatidão o que causou tal modificação e as possíveis maneiras de recuperar o dado original.

Segundo Fayad; Schmidt, (1997) *frameworks* representam uma estrutura formada por blocos pré-fabricados de software em que os programadores podem usar, estender ou adaptar para uma solução específica e linguagens de padrões.

Para Teixeira; Cervi, (2010) é comum que usuários utilizando sistemas corporativos compartilhem funções e módulos executando operações de inclusão, alteração ou exclusão em conjuntos de dados semelhantes. Desta forma, a sobreposição da informação alterada simultaneamente por dois usuários em um mesmo registro é um risco próximo à realidade dos sistemas transacionais. Para estes sistemas, tão importante quanto conseguir mesclar os dados de duas alterações distintas evitando assim a sobreposição de uma das transações, é que o sistema transacional consiga identificar ao administrador as operações ocorridas na base de dados e como atuar de forma corretiva no problema operacional.

O objetivo deste trabalho é apresentar uma *framework* de auditoria para projetos desenvolvidos na linguagem de programação Java, junto com o auxílio de uma *framework* de persistência, o Hibernate. Esta *framework* é integrada ao projeto com baixo acoplamento e total transparência para o desenvolvedor da aplicação. O estudo de caso demonstra o log e recuperação das transações são efetuadas com o mínimo de complexidade para o usuário administrador. A relevância deste trabalho está associada a evolução de ferramentas automatizadas para recorrentes problemas nas equipes de desenvolvimento de sistemas transacionais.

2. MATERIAIS E MÉTODOS

2.1 Marco Teórico

Simon; Szeliga, (2007) abordam a auditoria por análise de *streams* (dados em tempo real) e logs (registros armazenados pelo sistema). Eles descrevem tanto o aumento dos SGBD's quanto o crescente uso do mesmo no decorrer do tempo, além da evolução e expansão. Em paralelo, a necessidade de auditoria dos dados ganhou atenção. Os autores utilizaram como base o banco de dados PostgreSQL, este, que é um sistema gerenciador

de banco de dados relacional, que conta com o suporte à *triggers* (função executada sempre que um evento específico ocorre), funções, *procedures* (procedimentos, ou coleção de instruções) e gerenciamento de diversos tipos de dados, permitindo assim uma maior customização de logs a serem gerados. Desta forma para realização da leitura desses logs foram empregados o Borealis, um sistema distribuído para processamento de streams. Sendo assim, os dados, antes de serem armazenados são auditados, fazendo com que o seu uso, esteja ligado a sistemas que demandem de rápidas respostas, fazendo com que a análise seja feita em tempo real, tornando-se extremamente importante para padrões que precisem ser identificados em um grande fluxo de dados.

Para Garcia et al. (2008) o problema crucial, é identificar quem ou o que ocasionou algo em um sistema de banco de dados. Para isso, os autores abordam como solução o uso de trilhas de auditoria, apresentando de forma sucinta os conceitos e a problemática sobre o uso da mesma. Explicam a dificuldade de se projetar uma solução, pois há a necessidade de análise de um leque de ações que devem ser registradas, por exemplo, como será feita a análise da trilha, como será armazenada a trilha, o que fazer quando não houver mais espaço para registro na trilha, entre outros. Os autores citam que se ter um “rastro” de tudo que o é feito no sistema, como, inclusão, alteração e exclusão, é um recurso valioso para qualquer administrador de bancos de dados.

Teixeira; Cervi, (2010) apresentam a ferramenta "Audi-DML", que realiza auditoria usando *Data Manipulation Language* (DML do inglês Linguagem de Manipulação de Dados) e triggers, que são funcionais em SGBD's Oracle e Firebird. Posteriormente à apresentação, os autores demonstraram algumas funcionalidades desta ferramenta, realizando também alguns testes como inserção, alteração e exclusão em um sistema automatizado. No Oracle, foi realizado o teste com, e sem, a auditoria da "Audi-DML". No Firebird o teste foi auditado com o "Audi-DML" e também com a solução comercial "IB LogManager". Após a realização dos testes os autores perceberam uma queda no desempenho com o sistema que estava sendo auditado, porém, a maior diferença está no arquivo gerado pelo sistema auditor. No Firebird, o "Audi-DML" gerou 29.190.900 (vinte e nove milhões, cento e noventa mil e novecentos) registros na tabela auxiliar de auditoria. E para o mesmo teste realizado com o "IB LogManager" foi gerado 97.303.000 (noventa e sete milhões, trezentos e três mil) registros na tabela referente aos dados dos campos alterados, e mais 29.190.000 (vinte e nove milhões, cento e noventa mil) registros na tabela de dados adicionais da operação (implementado pelo IB LogManager), totalizando assim 126.493.000 (cento e vinte e seis milhões, quatrocentos e noventa e três mil) registros da auditoria.

Para Kraemer; Vogt, (2004) o Hibernate é um serviço de consulta e persistência Objeto-Relacional para Java, que faz os registros contidos banco de dados serem transformados em objetos, e as informações destes objetos em Java, serem persistidos em forma de linha e coluna. Para que isto aconteça, eles descrevem a sua arquitetura, as bibliotecas que a compõe, a sua fácil instalação e as maneiras de se configurar através dos arquivos da classe de configuração (*Configurations*). Os autores ainda criam uma Fábrica de Sessões através do objeto de configuração e arquivos XML que abordam o Mapeamento Objeto-Relacional básico, com uma tabela comparando os tipos de dados no Hibernate, na classe Java e o tipo equivalente no Banco de Dados. Por fim, demonstra-se como é feita a manipulação de objetos persistentes propondo uma comparação com outros *frameworks* de persistência, tendo como requisitos: “top-down”, “bottom-up”, “licença” e outros.

Paes; Fernandes; Azambuja, (2007) caracterizam a metodologia de orientação a aspectos, que foi desenvolvida em 1996 e pode ser utilizada com a Programação Orientada a Objetos (P.O.O.). Os autores abordam as estruturas de aspectos (aspects) que devem ser vistas como um componente que contém uma estrutura de pontos de corte (pointCuts) e citaram os pontos de junção (joinPoints) que constituem um parâmetro bem definido no fluxo de execução interrompendo o seu curso natural para inserir novas funções ou rotinas. Os pontos de corte, assemelham-se aos métodos de classe em Java, e os avisos (Advices), que são trechos de códigos associados a pointCuts, que introduzem um novo comportamento em todos os joinPoints representados pelo pointCut, além de listaram os tipos de caracteres especiais, tipos de designadores de pointcuts e tipos de avisos. Assim os autores desenvolveram duas modelagens como experimento, para um sistema de controle bancário. Uma das modelagens foi desenvolvida com orientação a objetos e a outra com orientação a aspectos. O experimento mostrou que foi possível concentrar a preocupação de registrar os logs das movimentações efetuadas em apenas um local. Comprovando que através da programação orientada a aspectos foi possível reduzir a complexidade do código fonte, tornando-o mais simples e de maior entendimento.

2.2 Metodologia

A *framework* de auditoria foi arquitetada para salvar os logs das suas auditorias de três maneiras:

- Bancos de dados;
- Arquivos Texto;
- Arquivos XML (*eXtensible Markup Language* ou linguagem de marcação);

A integração entre a *framework* de auditoria e a aplicação cliente é feita através de inserção das classes empacotadas em arquivos .jar (Java Archive) no ClassPath da aplicação. Após esta etapa toda vez que um evento reconhecido pela *framework* é disparado este é capturado através da interceptação dos eventos, providos pela implementação do aspecto, e manipulados para a persistência do log na auditoria.

Para que a *framework* saiba exatamente quais ações estão sendo executadas no banco, ela possui uma integração com o Hibernate, pois o mesmo, provê todas as informações necessárias para que a comunicação com o banco de dados seja bem-sucedida. O Hibernate é o responsável por toda ação executada dentro do banco, sendo assim, a *framework* utiliza essa integração para auditar as ações e valores do banco de dados. Os valores do banco são obtidos através de uma funcionalidade nativa do Java, o reflection, que permite que o Java faça uma introspecção, ou seja, examine suas propriedades e estruturas a fim de obter os valores de todos os atributos de uma determinada classe.

O cliente tem a liberdade de persistir suas informações em qualquer base de dados disponível para acesso via Hibernate, onde uma das grandes ajudas que ele provê ao sistema é a possibilidade de persistir os dados em qualquer base disponível para acesso via SQL. Pelo fato do Hibernate trabalhar com qualquer base de dados, há uma maior abrangência dos diversos bacos de dados utilizados em aplicações web quanto *desktop* como por exemplo: MySql, SqlServer, Oracle. A Figura 1 representa a arquitetura básica definida para esta *framework*.

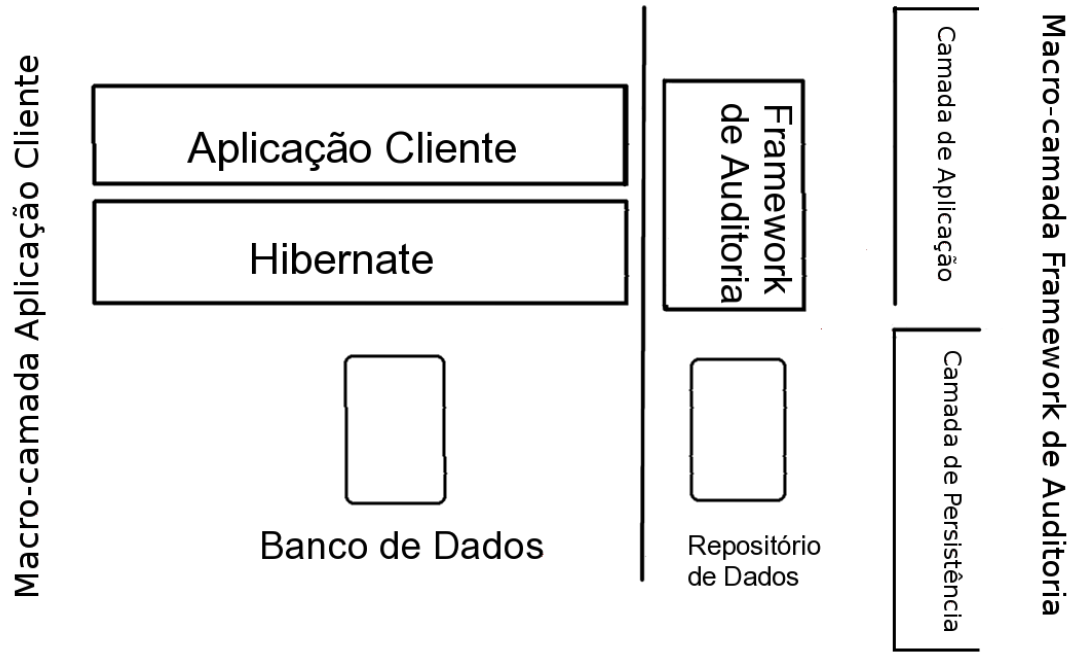


Figura 1: Arquitetura básica da *Framework* de Auditoria.
Fonte: Os autores.

2.2.1. Macro-Camada da Aplicação-Cliente

A camada de Aplicação Cliente é representada pelo grupo de aplicações que o usuário utiliza para ter acesso à base de dados. Estas são desenvolvidas em Java e ficam alocadas no mesmo servidor físico, facilitando a sua integração com o *framework* de Auditoria. Na prefeitura municipal de Cachoeiro de Itapemirim - ES, cada setor tem sua aplicação específica, pois cada uma dessas tem suas próprias regras de negócio e suas tabelas, onde estas, se comunicam diretamente com a camada do Hibernate.

A camada Hibernate é onde estão definidas todas regras de conexão com o banco de dados (nome do banco, usuário, senha, endereço do banco, e demais configurações relacionadas) e quais são as classes da aplicação estão mapeadas, através do uso do Mapeamento Objeto-Relacional. Sem a camada Hibernate não há nenhuma comunicação com o banco de dados.

Banco de Dados é camada em que todos os dados utilizados pelas aplicações clientes serão persistidos. Algumas tabelas do banco, são compartilhadas entre as várias aplicações clientes e outras, tem seu acesso restringido a determinadas aplicações.

2.2.2. Macro-Camada *Framework* de Auditoria

A camada do f*ramework* de Auditoria é executada em paralelo as camadas de aplicação-cliente e do Hibernate. Ela que é a responsável em persistir os dados necessários para a análise e auditoria dos registros do sistema. Tendo seus métodos chamados pelo AspectJ, assim quando qualquer aplicação precisar realizar uma conexão com o Banco de Dados, a f*ramework* realizará à auditoria utilizando os dados contidos nas aplicações-clientes e os registros do Banco de Dados.

Na camada de repositório de dados, encontra-se os registros já auditados. Estes estão sendo persistidos no banco de dados, em arquivos XML e em arquivos de texto simples, onde cada tipo de informação é armazenada em um tipo de base diferente. É no Banco de Dados que são armazenados as persistências dos dados, pois é lá, que estão os registos de tabelas. Já os dados que são persistidos em XML e nos arquivos simples de texto, são referentes aos erros que ocorrem no sistema, exceções e observações. Somente o f*ramework* de auditoria possui acesso ao repositório de dados.

2.2.3. Diagrama de Classes

A Figura 2 representa o modelo do Diagrama de Classes do sistema. Onde a classe EntityManager está diretamente ligada à classe AdMovi, e a classe Session está ligada a EntityManager via AspectJ.

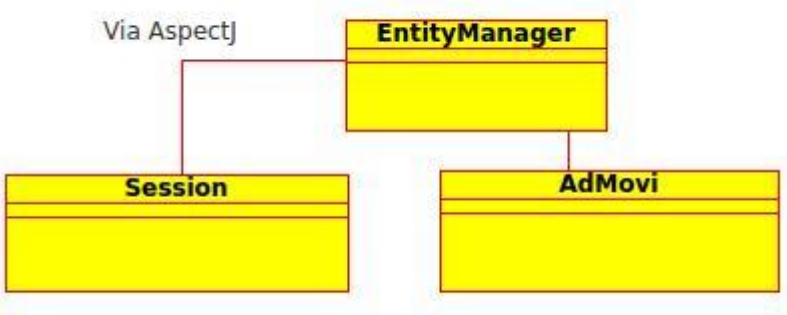


Figura 2: Modelo do Diagrama de Classes.
Fonte: Os autores.

Para isto, o primeiro passo é modelar o domínio da aplicação. O projeto foi concebido para ser o mais minimalista e simplificado possível sem perder performance e a qualidade dos dados auditados. A Tabela 1 identifica a entidade de auditoria e seus respectivos atributos:

Tabela 01 – Entidades de auditoria e seus atributos.

Nome	Descrição	Tipo	Obrigatório
moviCodigo	Identificador único do registro	Long	SIM
userLogin	Login ou nome do usuário que realizou a ação	String	SIM
moviData	Data e hora da ação	Date	SIM
moviClasse	Classe de entidade que sofreu a ação	String	SIM
moviAcao	Ação realizada no sistema (Insert, Update, Delete, Login, Error)	Char	SIM
moviPk	Chave primária do registro auditado	String	SIM
moviLog	Registra eventos de erros e observações do registro gravado.	String	NÃO
moviCampos	Valores dos atributos serializados da classe que sofreu a ação.	String	NÃO

Fonte: Os autores

Essa classe trabalhará junto com o AspectJ para a realização da auditoria. Assim que as funções do AspectJ iniciarem, elas faram a chamada das funções da classe Java passando os valores dos respectivos atributos.

Para a auditoria, foi escolhido guardar as seguintes informações: O usuário; A data da alteração; A tabela alterada; A operação realizada (inclusão, alteração, exclusão, erro ou login); A chave primária (*Primary Key*) do registro; Um log de erro ou observações da movimentação; E os campos dos registros auditados.

A Figura 3 demonstra a implementação da classe Java Admovi, que é a responsável pela persistência das informações necessárias à auditoria.

```
1  public class Admovi implements Serializable {
2      private static final long serialVersionUID = 1L;
3      @Id
4      private Integer movicodigo;
5      private String userlogin;
6      private Date movidata;
7      private String movitabela;
8      private String movioperacao;
9      private String movipk;
10     private String moviaplicacao;
11     private String movilog;
12     private String movicampos;
13     ...
14     /* Getters and  Setters methods */
15 …}
16
```

Figura 3: Implementação da classe Java Admovi.
Fonte: Os autores

Para maior segurança do sistema, todos os atributos do sistema estão sendo declarados como privado (*private*), o que restringe o acesso aos mesmos somente a classe em que eles são declarados. Desta forma, eles só podem ser acessados de fora da classe através do métodos “*get*” (leitura) e “*set*” (gravação).

Está sendo utilizando o Hibernate para realização do mapeamento da tabela. Observando logo acima do atributo “moviCodigo” temos a tag “@Id”, ela é identifica como a chave primária do dado a ser persistido e é do tipo inteiro (*Integer*). Esse valor será único para cada linha auditada.

O atributo “userLogin” é referente ao usuário a ser auditado, e é do tipo String (tipo primitivo de dados que armazena texto), caso possua letras no login do usuário . O atributo

“moviData” é do tipo Date (tipo primitivo para data) que armazenará a data e a hora que o usuário fez a alteração na tabela que está sendo auditada.

Para persistir o tipo de movimentação realizada no banco auditado, é utilizado o atributo, também do tipo String, chamado de “moviOperacao”, este poderá conter o valor “I” referente a inclusão (*insert*), “U” alteração (*update*), “D” exclusão (*delete*), “E” erro ou “L” login.

O atributo responsável por persistir o valor da chave primária (pk) da tabela auditada é o atributo “moviPk”, também do tipo String. Como esse valor é o da chave primária da tabela auditada, ela pode se repetir dentro da tabela gerada para auditoria, pois o valor gerado para chaves primárias não pode ser repetido. Desta forma o atributo de identificação da tabela de auditoria é adicionado pelo atributo “moviCodigo” (já especificado anteriormente). A intenção é facilitar as pesquisas e recuperação de informação.

Como várias aplicações podem estar acessando a mesma base de dados, o atributo “moviAplicacao” persistirá a informação sobre qual a aplicação fez a modificação na base de dados. Ex: A prefeitura de Cachoeiro de Itapemirim - ES, dispõe de uma grande base de dados para persistir informações dos moradores da cidade. Sendo dados são utilizados por vários setores, acidentalmente, um usuário da secretaria de educação alterou o nome de um cidadão nessa base de dados, e a secretaria de saúde precisa encontrar esse cidadão, porém não o encontra porque o nome dele foi alterado no sistema. Dessa maneira, o atributo “moviAplicacao” irá ter persistido que a alteração (*update*) aconteceu pela aplicação do sistema da secretaria de educação.

As observações e os erros gerados dentro do banco de dados também precisam ser auditados para uma posterior análise e correção, assim é utilizado o atributo “moviLog”, que é do tipo String.

Os valores a serem auditados referentes a erros e observações das execuções do banco de dados são persistidos através do atributo “moviCampos”, também do tipo String. Eles são serializados e armazenados em xml que irá ser persistido o valor após a modificação.

3. RESULTADOS E DISCUSSÃO

Comprovando a eficiência da f*ramework* de auditoria, foram realizados testes junto à Empresa DATACI (Companhia de Tecnologia da Informação de Cachoeiro de Itapemirim), que é responsável pelos sistemas da Prefeitura Municipal de Cachoeiro de Itapemirim. Foram utilizados o banco de dados Oracle 12g e o sistema Escol@r, desenvolvido pela

DATACI e ambos já em pleno funcionamento. O sistema Escol@r é o sistema responsável pela gestão acadêmica das escolas de Cachoeiro de Itapemirim, juntamente com todas as informações necessárias para a gestão pública direcionada à educação.

Na realização dos testes, utilizou-se de alguns comandos básicos e rotineiros como a inserção, alteração e exclusão de valores relacionado a alunos. Levando em consideração que o sistema já está em uso, possuindo uma grande quantidade de dados persistido no banco, ressalta-se que para a realização dos experimentos não foram utilizados nomes reais, não comprometendo o funcionamento do sistema.

O sistema Escol@r, assim como os demais sistemas disponibilizados pela DATACI, são providos via web, sendo obrigatório o uso de um login e senha em uma página principal, onde está disponibiliza todos os sistemas inclusos. Esse procedimento é extremamente importante pois há a necessidade de saber qual usuário executou cada ação no sistema. A Figura 4 mostra a tela de login do sistema responsável pelo acesso.

Figura 4: Tela de login do Sistema.
Fonte: Os autores

No momento que o usuário é autenticado e passa a ter acesso as funcionalidades do sistema, a *framework* de auditoria começa a ser executada em segundo plano, sem que o usuário perceba. Na realização dos experimentos foi adicionado o aluno “Aluno de Teste Para o Sistema de Auditoria” com a sua data de nascimento sendo salva como “13/06/2013”, como demonstrado pela Figura 5 a seguir.

Figura 5: Inclusão dos dados do Aluno de Teste.

Fonte: Os autores

A partir do instante que o sistema persistiu os dados referentes a esse aluno, a *framework* de auditoria também terminou de auditar esse registro, salvando todos os dados necessários para a identificação, tudo que foi feito e quem realizou a inserção. Para acessarmos a tela de consulta da *framework*, foi necessário executá-la diretamente na IDE de Desenvolvimento Eclipse, demonstrando que somente o desenvolvedor da ferramenta possui acesso no presente momento. Como o experimento está sendo controlado do início até o fim, temos a chave primária do aluno desde o momento de sua inserção, está é disponibilizada pelo sistema, caso haja a solicitação de auditar informações pertinentes a ele.

Após executar a *framework*, foi passado como parâmetro a chave primária do aluno e o sistema que foi auditado. A Figura 6 mostra o que foi retornado pelo sistema.

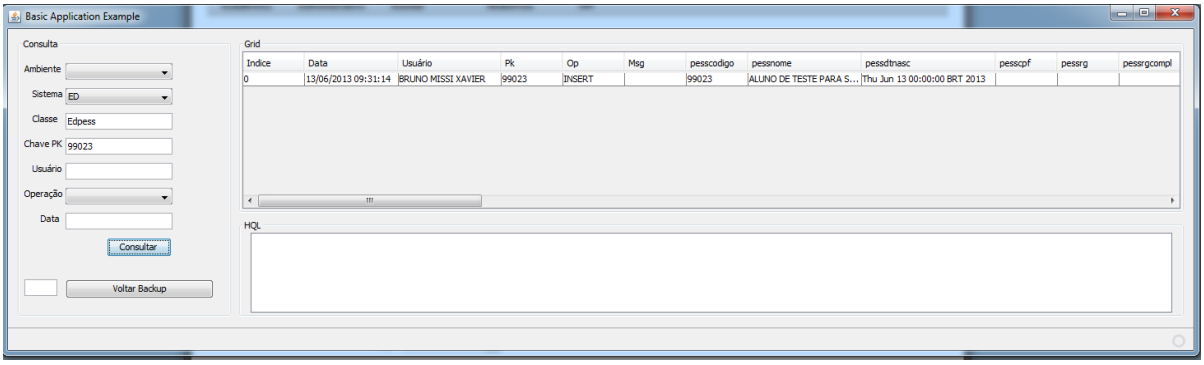


Figura 6: Auditoria da inserção do Aluno Teste.
Fonte: Os autores

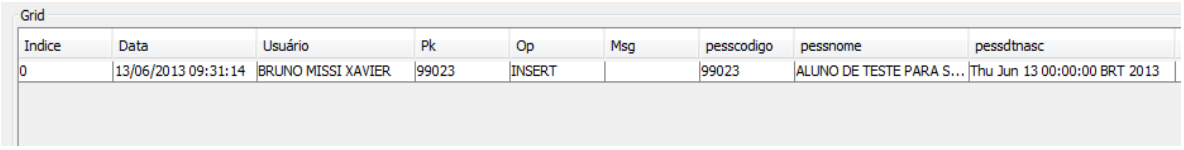


Figura 7: Zoom nos dados gerais da Auditoria.
Fonte: Os autores

Como é observado na imagem, o sistema disponibilizou de forma clara e organizada os valores referentes a execução da inserção do aluno, apresentando os valores dos campos referentes a data e hora, o usuário, a chave primária do registro auditado, o nome completo do aluno e sua data de nascimento.

Dando sequência, o usuário fez uma alteração (*update*) no nome do aluno acrescentando o valor “1 Alteração” no final do nome e manteve os demais valores intactos, e logo após persistindo os mesmos no banco de dados, conforme mostra a Figura 8.

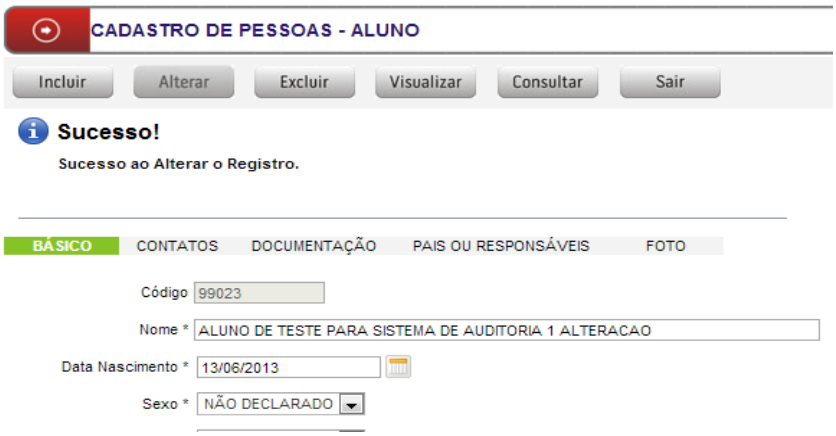
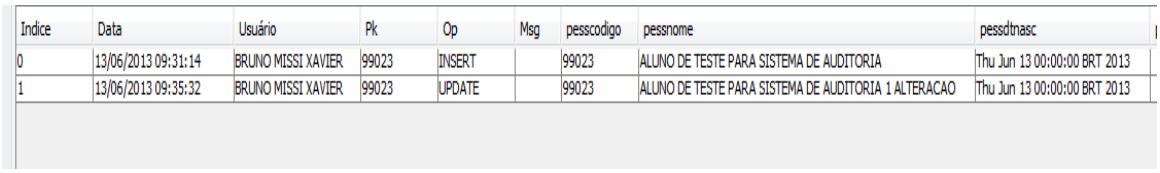


Figura 8: Alteração do nome do aluno.
Fonte: Os autores, 2013.

Como a *framework* foi desenvolvida para auditar todas ações que são executadas no banco de dados, o registro da alteração foi auditado e manteve a legibilidade na leitura dos registros. Sempre que ocorrer uma nova execução no sistema e um valor for auditado, o valor anterior a ele permanecerá intacto e listado de forma cronológica para facilitar a análise, assim como é demonstrado na Figura 9.



Indice	Data	Usuário	Pk	Op	Msg	pesscodigo	pessnome	pessdtnasc
0	13/06/2013 09:31:14	BRUNO MISSI XAVIER	99023	INSERT		99023	ALUNO DE TESTE PARA SISTEMA DE AUDITORIA	Thu Jun 13 00:00:00 BRT 2013
1	13/06/2013 09:35:32	BRUNO MISSI XAVIER	99023	UPDATE		99023	ALUNO DE TESTE PARA SISTEMA DE AUDITORIA 1 ALTERACAO	Thu Jun 13 00:00:00 BRT 2013

Figura 9: Exibição dos dados auditados.

Fonte: Os autores

Como mostrado na Figura 9, o campo relacionado ao nome do aluno contém o valor antigo e o valor atualizado após a modificação feita pelo usuário. Facilitando assim a comparação e a recuperação dos valores antigos. O sistema não possui limites de audições por tipo de execução, como por exemplo: Podem ocorrer inúmeras alterações nos dados do “Aluno de Teste”, que a auditoria persistirá todas as modificações. Para comprovar essa afirmativa, foi realizada outra operação de alteração dos dados, como mostra a Figura 10 e na Figura 11 novamente são mostrados os registros auditados.

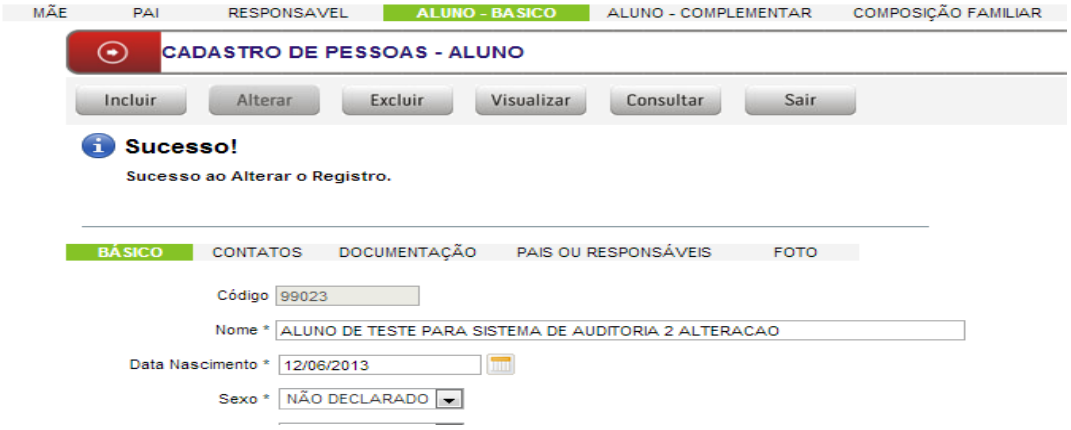
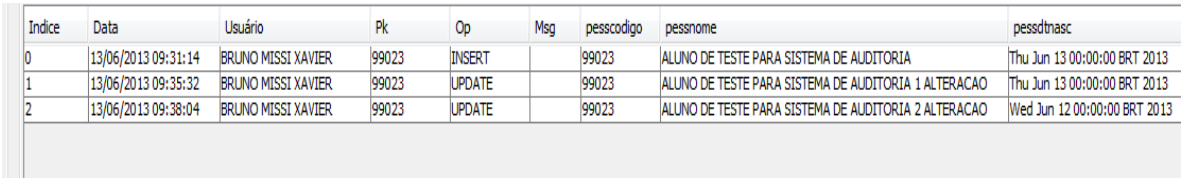


Figura 10: Segunda alteração no Aluno de Teste.

Fonte: Os autores



Indice	Data	Usuário	Pk	Op	Msg	pesscodigo	pessnome	pessdtnasc
0	13/06/2013 09:31:14	BRUNO MISSI XAVIER	99023	INSERT		99023	ALUNO DE TESTE PARA SISTEMA DE AUDITORIA	Thu Jun 13 00:00:00 BRT 2013
1	13/06/2013 09:35:32	BRUNO MISSI XAVIER	99023	UPDATE		99023	ALUNO DE TESTE PARA SISTEMA DE AUDITORIA 1 ALTERACAO	Thu Jun 13 00:00:00 BRT 2013
2	13/06/2013 09:38:04	BRUNO MISSI XAVIER	99023	UPDATE		99023	ALUNO DE TESTE PARA SISTEMA DE AUDITORIA 2 ALTERACAO	Wed Jun 12 00:00:00 BRT 2013

Figura 11: Análise dos registros auditados.

Fonte: Os autores

Como é apresentado na Figura 11, pode se observar que o nome do aluno foi alterado de “1 Alteração” para “2 Alteração” e o campo relacionado a data de nascimento foi

modificado de “13/06/2013” para “12/06/2013”. Essa modificação foi feita às “09:38:04” no dia “13/06/2013” pelo usuário “Bruno”.

A fim de simular um ambiente real, onde imprevistos podem acontecer, simulamos a exclusão dos dados do “Aluno de Teste”. Normalmente, após uma exclusão, sua recuperação tona-se um pouco mais complicada de ser realizada, sendo necessário recorrer a backups antigos com o último valor persistido. A Figura 12 mostra que o dado foi realmente excluído da base de dados original.

Figura 12: Confirmação da exclusão.
Fonte: Os autores

Já que é função da f*ramework* de auditoria, os dados que existiam do “Aluno de Teste” na hora da exclusão são todos persistidos, seguindo a mesma forma de exibição para a análise e auditoria conforme podemos observar na Figura 13.

Indice	Data	Usuário	Pk	Op	Msg	pesscodigo	pessnome	pessdtnasc	pe
0	13/06/2013 09:31:14	BRUNO MISSI XAVIER	99023	INSERT		99023	ALUNO DE TESTE PARA SISTEMA DE AUDITORIA	Thu Jun 13 00:00:00 BRT 2013	
1	13/06/2013 09:35:32	BRUNO MISSI XAVIER	99023	UPDATE		99023	ALUNO DE TESTE PARA SISTEMA DE AUDITORIA 1 ALTERACAO	Thu Jun 13 00:00:00 BRT 2013	
2	13/06/2013 09:38:04	BRUNO MISSI XAVIER	99023	UPDATE		99023	ALUNO DE TESTE PARA SISTEMA DE AUDITORIA 2 ALTERACAO	Wed Jun 12 00:00:00 BRT 2013	
3	13/06/2013 09:40:28	BRUNO MISSI XAVIER	99023	DELETE		99023	ALUNO DE TESTE PARA SISTEMA DE AUDITORIA 2 ALTERACAO	Wed Jun 12 00:00:00 BRT 2013	

Figura 13: Consulta de todas as operações realizadas no “Aluno de Teste”.
Fonte: Os autores

Graças a *framework* de Auditoria, foi possível, como consta na Figura 13, obter todo o ciclo de vida do “Aluno de Teste” dentro do sistema Escol@r. Onde todos os dados inseridos, alterados e excluídos foram persistidos em um repositório de dados que é específico para esse fim. E mesmo após um longo tempo após a exclusão, será de muita facilidade recuperar os dados de tudo que já foi realizado na base de dados.

4. CONSIDERAÇÕES FINAIS

Este trabalho apresentou uma proposta de arquitetura referente a uma *framework* de auditoria para sistemas desenvolvidos na tecnologia Java, integrados com o *framework* de Mapeamento Objeto-Relacional Hibernate. O desenvolvimento e implantação desta ferramenta, foi realizada na Companhia de Tecnologia da Informação de Cachoeiro de Itapemirim (DATACI), que hoje, usufrui desta estrutura de auditoria como parte vital dos seus sistemas Web.

Este recurso é utilizado amplamente pela empresa. Dentre as diferentes formas de uso, sua principal função é em reuniões com usuários, para auditar as ações dos mesmos dentro dos sistemas. Também proporciona à empresa DATACI, maior confiabilidade na gestão e manutenção dos serviços (sistemas) prestados à Prefeitura Municipal de Cachoeiro de Itapemirim (PMCI).

No momento que os membros da equipe entenderam, que esta ferramenta se torna uma parte vital do desenvolvimento de sistemas nesta tecnologia, se tornou prática que nenhum sistema seja confeccionado sem a inclusão da mesma.

Seu uso é de extrema eficácia e deve ser usado por toda empresa que preste serviços de base de dados e sistemas, porém há pouquíssimas ferramentas desse aspecto para uso, tanto comercial quanto livre. A *framework* evita que qualquer dado que seja, se perca, aumentando a integridade, segurança e confiabilidade do sistema e da base de dados.

A etapa de experimentos demonstrou a viabilidade de acoplamento desta *framework* em um sistema já existente e em produção. A partir deste momento, foi possível resgatar o ciclo de vida da informação gerada através da estrutura criada para armazenar os dados de auditoria.

REFERÊNCIAS

ELMASRI, Ramez; NAVATHE, Shamkant B. **Sistemas de Banco de dados: Fundamentos e Aplicações.** 3° Edição, Rio de Janeiro, Editora LTCRJ, 2002.

FAYAD, M. E.; SCHMIDT, D. C. **Object-oriented Application frameworks. Communications of the ACM.** Vol. 40, 10 p., 1997.

GARCIA, Cleber dos Santos et al. **Auditoria em Sistemas de Informação – Trilhas de Auditoria.** Brasília, 2008.

GIRON, Paulo Roberto. **Modelo de Informações Para a Auditoria de Sistemas de Gestão em Pequenas e Médias Empresas.** Florianópolis, 2003.

KRAEMER, Fabiano; VOGT, Jerônimo Jardel. Hibernate**, um Robusto Framework de Persistência Objeto-Relacional**. Porto Alegre, 2004.

PAES, Fabiano Gama; FERNANDES, Mikael de Souza; AZAMBUJA, Rogério Xavier de. **Utilizando a Programação Orientada a Aspectos na Implementação de um Registro de Log.** Santa Maria, 2007.

SIMON, Fernando; SZELIGA**,** Karla. **Auditoria em Banco de Dados Através da Análise de Streams**. Curitiba, 2007.

TEIXEIRA, Rogério Marros; CERVI, Cristiano Roberto. **Auditoria de Comandos DML em Bancos de Dados Oracle e Firebird**. Passo Fundo, 2010.

XAVIER, Bruno Missi; SILVA, Alcione Dias da; GOMES, Geórgia Regina Rodrigues. **Análise comparativa de algoritmos de redução de radicais e sua importância para a mineração de texto**. Rio de Janeiro, v.5, n.1, p. 84-99, 2013.

Sobre os Autores

Emanuel Bonfante Muniz: Aluno graduando do curso Análise e Desenvolvimento de Sistemas da IES Centro Universitário São Camilo - Espírito Santo. Atua na área de segurança em servidores Linux. E-mail: emanuel.ebm.ti@gmail.com.

Renato Sousa Botacim: Aluno graduando do curso de Sistemas de Informação da IES Centro Universitário São Camilo - Espírito Santo. Atua na área de desenvolvimento de software. E-mail: botacim.renato@gmail.com.

Bruno Missi Xavier: Possui graduação em Sistemas de Informação pela União Social Camiliana (ES)(2004) e mestrado em Pesquisa Operacional e Inteligência Computacional pela Universidade Candido Mendes(2012). Atualmente é Consultor Interno de Gestão da DATACI - Companhia de Tecnologia da Informação, Professor Horista do Centro Universitário São Camilo - ES, Professor Horista da Faculdade de Castelo e Tutor a Distância do Instituto Federal de Educação, Ciência e Tecnologia do Espírito Santo. Tem

experiência na área de Ciência da Computação, com ênfase em Sistemas de Computação. Atuando principalmente nos seguintes temas: Mineração de Texto, Descoberta de Conhecimento, Visão Computacional, Heurísticas Distribuídas.

Marcelo Schuster Albuquerque: Possui graduação em Tecnologia Em Processamento de Dados pelo Centro Universitário do Espírito Santo (1998), Pós Graduação Design Instrucional para Ensino à Distância, pelo IBDIN, Mestre em Pesquisa Operacional e Inteligência Computacional Universidade Cândido Mendes - Campos. Atualmente é Coordenador dos Cursos de Sistemas de Informação e de Tecnologia e Análise de Sistemas do Centro Universitário São Camilo, em Cachoeiro de Itapemirim. Tem experiência na área de Ciência da Computação, com ênfase em Redes de Computadores, Computação e Sociedade, Sistemas de Informação, Ensino à Distância e Inteligência Computacional.