Comunicado Técnico 124
ISSN 1677-8464
Dezembro, 2016
Campinas, SP

Fonte: Google Image

# Auditoria de entidades de banco de dados com o Hibernate Envers

Daniel Rodrigo de Freitas Apolinário[1]
Isaque Vacari[2]
Leonardo Ribeiro Queiros[3]

## Introdução

O chamado dilúvio de dados ocorrido com o progresso exponencial da tecnologia da informação tem atingido todas as áreas da vida humana. Com o aumento do volume de dados que são coletados, processados e armazenados, também cresce a preocupação com a segurança da informação, já que dados precisam ser protegidos devido ao seu valor, quer seja econômico ou de outra natureza. A segurança da informação é um tema muito amplo que envolve tecnologia, processos, pessoas, leis e normas. Devido à sua amplitude, a segurança possui várias formas e vários níveis de implementação. A auditoria é um dos componentes de segurança e responsável por atacar algumas questões relacionadas ao tema. Não há segurança sem auditoria, e não há necessidade de auditoria sem a necessidade de segurança (BEN-NATAN, 2005). Portanto, segurança e auditoria devem ser implementadas de maneira integrada (YANG, 2009).

Na computação, dados costumam ser armazenados em estruturas que são chamadas de bancos de dados. Portanto, a auditoria de banco de dados é uma das várias formas de auditoria utilizadas como parte da estratégia de segurança da informação. Como um banco de dados é muito rico em funcionalidades, diferentes tipos de trilhas de auditoria podem ser produzidas para este ambiente (BEN-NATAN, 2005).Todas as operações que podem ser realizadas em um banco de dados quer seja por um administrador, um usuário ou uma aplicação que possui acesso aos dados são passíveis de serem auditadas e a definição do tipo mais apropriado de auditoria requer uma análise que varia de acordo com o requisito existente, ou seja, o que se deseja assegurar com relação aos dados a serem auditados.

Um dos tipos de auditoria de banco de dados muito comum na maioria das iniciativas é a auditoria de mudanças em dados sensíveis (BEN-NATAN, 2005). Esse tipo de auditoria também pode ser chamado de auditoria de atividade Data Manipulation Language (DML) e envolve o registro de cada alteração realizada em um dado (que pode ser uma inclusão, uma atualização ou uma exclusão). Este registro pode conter informações de quem realizou, o que foi alterado e quando a operação foi realizada. Os valores antigos e os novos também podem ser armazenados pelo mecanismo de auditoria de modo a prover um conjunto de informações relevantes para rastrear com precisão as mudanças realizadas nos dados. Este tipo de auditoria é importante para garantir requisitos como integridade, conformidade e confiabilidade dos dados e sistemas que os gerenciam, pois, por meio destes registros podem ser identificadas mani-

[1] Cientista da computação, especialista em Plataformas de Desenvolvimento Web, analista da Embrapa Informática Agropecuária, Campinas, SP.
[2] Tecnólogo em Processamento de Dados, mestre em Ciência da Computação, analista da Embrapa Informática Agropecuária, Campinas, SP.
[3] Cientista da computação, doutor em Engenharia Agrícola, analista da Embrapa Informática Agropecuária, Campinas, SP.

pulações incorretas e/ou indevidas nos dados e este histórico também possibilita sua reversão e correção. Uma vez que é identificada este tipo de manipulação, os registros de auditoria capacitam a análise do fator causador do problema, isto é, se houve alguma intrusão, mau uso ou um bug de sistema.

Alguns sistemas gerenciadores de banco de dados como *Oracle* e *Microsoft SQL Server* possuem suporte nativo a este tipo de auditoria. Outros como *PostgreSQL* possuem extensões que executam esta função. Também temos ferramentas/bibliotecas independentes dos softwares gerenciadores de banco de dados que se propõem a realizar auditoria em mudanças de dados. O uso de uma ou de outra abordagem ou ferramenta depende de uma análise de custo/benefício levando-se em consideração os requisitos da auditoria e os recursos disponíveis para a sua implementação.

Este trabalho apresenta o uso da ferramenta Hibernate Envers (HIBERNATE ENVERS, 2016) como uma opção para a auditoria de entidades de bancos de dados.

Com o objetivo de ajudar a compreensão do leitor, este trabalho está dividido em 3 seções. Na primeira seção, o Hbernate *Envers* é apresentado com suas funcionalidades e com uma visão

## Apresentação do Hibernate Envers

O *Hibernate* (HIBERNATE, 2016) é uma ferramenta de *Object-Relational Mapping* (ORM) escrita em Java. Este tipo de ferramenta tem como principal característica permitir que seja feito de maneira simples o mapeamento entre o modelo relacional de um banco de dados e o modelo de objetos de um sistema. Juntamente com o mapeamento, este tipo de ferramenta provê uma camada de abstração para acesso aos dados fazendo com que a camada de aplicação consiga acessar estes serviços por meio de funções de mais alto nível sem se preocupar com os detalhes da conexão aos bancos de dados. O mapeamento de entidades é simples e feito via arquivo de configuração XML ou por anotações nas classes que representam as entidades.

O *Envers* é um módulo da ferramenta *Hibernate* que tem como objetivo permitir de maneira fácil a realização de auditoria e versionamento de classes persistentes, ou seja, entidades de banco de dados mapeadas para o modelo de classes de uma aplicação (HIBERNATE ENVERS, 2016)
Algumas das principais funcionalidades fornecidas pelo *Envers* estão listadas a seguir:

- Permite a auditoria de todas as entidades mapeadas de acordo com a especificação JPA (Java Persistente API).
- Permite a auditoria de alguns mapeamentos específicos do Hibernate, como tipos customizados e coleações de tipos de dados simples como strings, inteiros etc.
- Log de dados para cada revisão pelo uso de uma entidade “revisão”. Este log contém a informação de como o dado “era”
- antes de uma alteração e como ele “ficou” depois da alteração.
- Classes e métodos específicos para consultas a dados históricos de auditoria.

O *Envers* trabalha com o conceito de revisões que já é muito utilizado em sistemas de controle de versão. Ou seja, para cada transação realizada em um banco de dados é criada uma nova revisão e todas as mudanças realizadas em entidades auditáveis estarão associadas a um número único que identifica esta revisão.

### Visão geral do funcionamento da ferramenta

Como o *Envers* funciona com base no mecanismo de persistência do próprio Hibernate para prover as funcionalidades de auditoria nas entidades de um modelo de classes, as alterações em código na aplicação são muito simples. Para cada entidade que se deseja auditar deve ser inserida uma anotação na própria classe para indicar que ela será auditada. Esta anotação é escrita com o texto “@Audited” antes da definição da classe. Existem configurações e customizações mais avançadas que são suportadas pelo *Envers*, mas esta anotação das classes é a configuração básica e suficiente para que uma aplicação já consiga sair usando o mecanismo de auditoria da ferramenta. Portanto, as alterações no código fonte da aplicação descritas acima para marcar quais são as entidades/atributos auditáveis são o primeiro passo para o funcionamento do *Envers*.

A Figura 1 exemplifica a facilidade da configuração das entidades a serem auditadas. No caso da entidade Pessoa usada como um exemplo fictício, basta adicionar uma linha com a anotação @Audited antes da declaração do nome da classe para que a partir deste momento, o *Envers* já “entenda” que os dados referentes a esta entidade devem ser auditados. Quando houver algum atributo específico que não se deseja auditar como um que seja do tipo blob, basta inserir

a anotação @NotAudited para o atributo. Também é possível usar a anotação @Audited(targetAuditMode=RelationTargetAuditMode.NOT_AUDITED) para os casos de atributos de relacionamentos com tabelas de domínio (relacionamentos do tipo muitos para um) indicando que estas tabelas não sofrerão alterações e por isso não precisam de auditoria. No exemplo da Figura 1 é exibido apenas uma parte da classe que representa a entidade que é o local onde é inserido a anotação do *Envers*. As anotações @Entity e @Table são da especificação JPA (Java Persistence API) utilizada neste exemplo.

```
import org.hibernate.envers.Audited;
import org.hibernate.envers.NotAudited;
import org.hibernate.envers.RelationTargetAuditMode;


@Entity
@Table(name="pessoa")
@Audited
public class Pessoa implements Serializable {
    ...
```

**Figura 1**. Trecho de código fonte de classe configurada para auditoria do Hibernate Envers.

O esquema de banco de dados do *Envers* é composto por uma tabela para armazenar as revisões. Esta tabela contém duas colunas essenciais: *id* e *timestamp*. Toda vez que houver um *commit* de uma transação no banco de dados, todas as alterações de dados contidas nesta transação estarão associadas a um único *id* de revisão. Além desta tabela que contém a revisão global, para cada entidade auditável do banco de dados existirá uma tabela de auditoria que recebe o nome da tabela original acrescido do sufixo "_aud" e que contém todos os campos da original acrescidos do campo referente ao id da revisão e de outro campo que armazenará o tipo da revisão (ADD = inclusão, MOD = modificação e DEL = remoção). Por último, é opcional que se utilize mais uma tabela para armazenar o nome da entidade que está sofrendo alguma alteração numa determinada revisão. No lado esquerdo da Figura 2, verifica-se um exemplo fictício da tabela pessoa pertencente a um modelo de dados utilizado por uma aplicação. Ainda neste exemplo, as três tabelas à direita com sufixo "_aud" representam o modelo de dados de auditoria. A tabela de revisao_aud possui a coluna usuario_id além das outras duas colunas obrigatórias mencionadas neste parágrafo. A Figura 2 também mostra que a tabela pessoa_aud possui os mesmos campos da tabela pessoa acrescidos dos campos revisao_id e tipo_revisao. A tabela revisao_entidade_aud é opcional e é responsável por armazenar quais são as entidades modificadas em cada revisão.

Para que o *Envers* consiga armazenar quaisquer outras informações na tabela que guarda as informações da revisão (como, por exemplo, adicionar a informação usuario_id na tabela revisao_aud do exemplo da Figura 2), é necessário a criação de uma classe anotada com *@RevisionEntity* que contenha todos os atributos necessários. Esta classe precisa conter os atributos obrigatórios *id* e *timestamp* além dos atributos que se deseja adicionar. No caso exemplificado, observa-se que o atributo usuario_id foi adicionado para armazenar quem foi o responsável por cada revisão no esquema (Figura 2) e também na classe (Figura 3). Qualquer outro atributo pode ser adicionado, como por exemplo o Internet Protocol (IP) da máquina do usuário ou qualquer outra informação que seja relevante para o requisito de auditoria. Um exemplo de uma classe que representa esta entidade da revisão contendo os atributos obrigatórios mais o atributo usuario_id está ilustrado na Figura 3. Esta figura também mostra que além dos campos já citados, também há um atributo do tipo *Collection* com o nome *modifiedEntityNames* que é responsável por indicar ao Hibernate que a tabela opcional para armazenar os nomes das entidades envolvidas em uma revisão será utilizada e a configuração deste atributo precisa conter as anotações

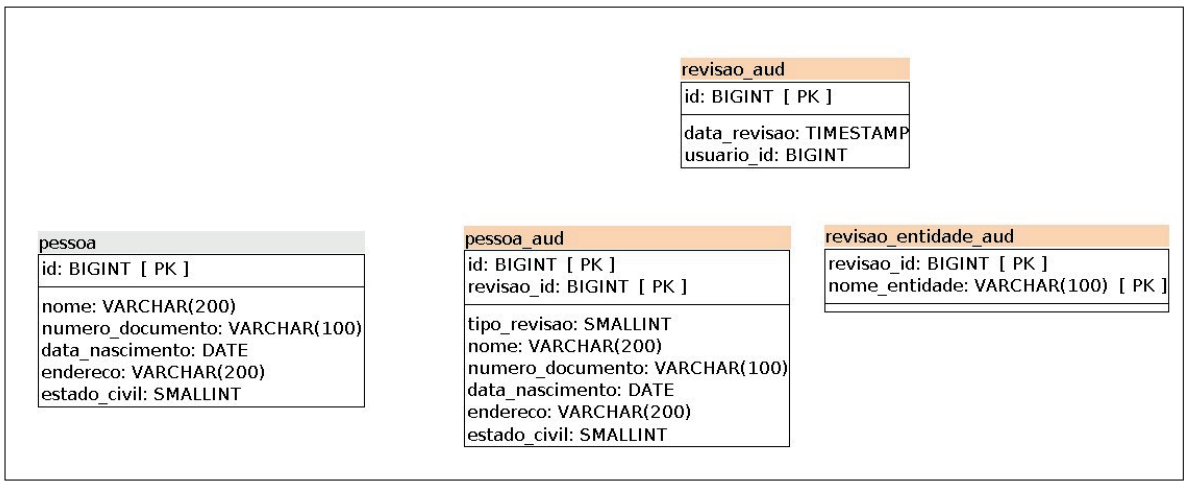


**Figura 2**. Exemplo de um modelo de dados de auditoria básico do *Envers* para uma tabela.

*@ElementCollection*, *@JoinTable* (no qual é indicado qual o nome da tabela que conterá as informações) e *@ModifiedEntityNames*.

```
@Entity(name="revisao_aud")
@Table(schema="auditoria_schema")
@RevisionEntity(MyRevisionListener.class)
public class MyRevisionEntity implements Serializable {

    @Id
    @GeneratedValue
    @RevisionNumber
    private long id;

    @RevisionTimestamp
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name="data_revisao")
    private Date dataRevisao;

    @Column(name="usuario_id")
    private long usuarioId;

    @ElementCollection
    @JoinTable(schema="auditoria_schema", name = "revisao_entidade_aud",
               joinColumns = @JoinColumn(name = "revisao_id"))
    @Column(name = "nome_entidade")
    @ModifiedEntityNames
    private Set<String> modifiedEntityNames;
    ...
```

**Figura 3**. Exemplo de customização da classe RevisionEntity.

Os valores dos atributos obrigatórios *id* e *timestamp* assim como o *modifiedEntityNames* são automaticamente preenchidos pelo *Envers* quando o mecanismo de auditoria é acionado, porém é necessário implementar o código que será executado para obter e preencher os atributos adicionais. Esta implementação é realizada ao criar uma classe que implementa a interface *org.hibernate.envers.RevisionListener* e sobrescrever o método *newRevision(...)* como é mostrado na Figura 4. O código exibido na Figura 4 ilustra a chamada a um serviço específico para a obtenção de um objeto do tipo Usuario e após o sucesso da obtenção desta informação o objeto *revisionEntity* que representa a tabela principal de revisão é alterado para incluir o código do usuário obtido.

O *Envers* também possibilita a realização de customizações por meio de ajustes no arquivo de configuração do Hibernate *persistence.xml*. Todas as possíveis configurações do *Envers* assim como a descrição do que significa cada uma delas estão disponíveis na sua documentação. Cada configuração do *Envers* tem um valor padrão que é assumido quando o usuário pode opte por não efetuar customização. No exemplo da Figura 5 são exibidos alguns exemplos simples de customizações que são possíveis, com valores configurados para as seguintes propriedades: *org.hibernate.envers.default_schema* que configura o nome do esquema

```
@ManagedBean
public class MyRevisionListener implements RevisionListener {

    @Override
    public void newRevision(Object revisionEntity) {
        MyRevisionEntity myRevEntity = (MyRevisionEntity) revisionEntity;
        Usuario usuario = null;
        try {
            MyService service = EJBUtils.lookup(MyServiceBean.class);
            usuario = service.getUsuario();
        } catch (IllegalArgumentException iaE) {
            ...
        }
        if (usuario != null && usuario.getId() != null) {
            myRevEntity.setUsuarioId(usuario.getId());
        }
    }
}
```

**Figura 4**. Implementação da classe RevisionListener para preenchimento do id do usuário.

do banco de dados de auditoria, *org.hibernate.envers.revision_field_name* que configura o nome da coluna obrigatória id na tabela que armazena informações da revisão e *org.hibernate.envers.revision_type_field_name* que configura o nome da coluna que identifica o tipo de revisão. Esta figura representa somente um trecho do arquivo de configuração do Hibernate no qual são configuradas as propriedades globais desta ferramenta.

Algumas funcionalidades básicas e customizações foram exemplificadas neste documento, porém existe uma gama de opções que não serão abordadas aqui tais como as classes específicas para consultas nos registros de auditoria, configuração de auditoria condicional, particionamento das tabelas de auditoria e outras que constam na documentação do *Envers* (HIBERNATE ENVERS, 2016). Esta documentação deve ser a referência para outros tipos de configurações que o usuário possa requisitar.

Portanto, esta visão geral do funcionamento pretende mostrar como a ferramenta *Envers* se propõe a resolver o problema da auditoria de entidades de bando de dados. Uma vez que  a ferramenta é instalada e configurada, automaticamente registra os dados de auditoria de acordo com as operações que são efetuadas no banco de dados. Estes registros de auditoria possibilitam o rastreamento das mudanças ocorridas

## Estudo de caso de uso: utilização do Hibernate Envers no projeto do SIExp

No âmbito do projeto MP5 “Gestão dos dados experimentais da Embrapa” foi desenvolvido o Sistema de Informação de Experimentos da Embrapa – SIExp. Este software tem a missão de ser uma ferramenta para dar suporte ao pesquisador da Embrapa no proces

```
<property name="org.hibernate.envers.default_schema" value="auditoria_schema"/>
<property name="org.hibernate.envers.revision_field_name" value="revisao_id"/>
<property name="org.hibernate.envers.revision_type_field_name" value="tipo_revisao"/>
```

**Figura 5**. Exemplo de customização de configurações do Hibernate no arquivo persistence.xml.

so de gestão de experimentos, contribuindo para algumas fases do ciclo de vida dos dados experimentais como planejamento, coleta, validação, descrição, preservação, descoberta e integração (DataONE, 2016).

O SIExp tem como objetivos iniciais principais a coleta e o armazenamento de dados coletados em experimentos, por isso a sua funcionalidade mais destacada é a de registro das avaliações de experimentos que pode ser feito por formulário Web, importação de planilha ou por dispositivo móvel. As avaliações são os dados brutos referentes às medições de todas as variáveis observadas nas unidades experimentais. Na sua primeira versão, ele contempla experimentos que utilizam delineamento estatístico para o sorteio de suas unidades experimentais. Um dos resultados mais importantes deste projeto foi o estabelecimento e implementação de uma norma de propriedade de acesso e de uso das informações experimentais que são armazenadas e gerenciadas pelo sistema. Esta norma propõe uma classificação dos dados de acordo com sua sensibilidade e define quem e quando pode ter acesso a quais tipos de dados.

Devido ao entendimento de que os dados de pesquisa da Embrapa armazenados pelo SIExp possuem alto valor científico e baseando-se nas questões de sensibilidade e de segurança destas informações, surgiu o requisito de auditoria dos dados do sistema para possibilitar a existência de um log de alterações, sendo possível assim identificar com precisão o usuário responsável por uma alteração, data e hora que foi efetuada e os valores antigos e novos. Munido deste tipo de ferramenta, o SIExp deseja prover mais confiabilidade e integridade dos dados que gerencia.

Assim, com relação ao aspecto técnico da implementação da auditoria dos dados, foram avaliadas algumas alternativas para a implementação deste requisito, tais como: utilização de um mecanismo de auditoria no próprio banco de dados através do uso de triggers; implementação na própria aplicação de um mecanismo de auditoria específico se utilizando de padrões de arquitetura para Web como *callback, interceptors* e *listeners*; utilização de algum *framework*/ferramenta de auditoria já existente e que fosse *open-source*. Das alternativas listadas acima, embora a primeira seja ainda muito utilizada, optou-se por não utilizar este tipo de mecanismo para separar as responsabilidades do banco de dados e também porque seria mais complexa a obtenção da identificação do usuário da aplicação no contexto do banco de dados e esta era uma informação essencial para ser usada na auditoria. A segunda alternativa foi descartada devido ao fato de que o refactoring de código fonte seria muito trabalhoso, a implementação não é trivial e também pela constatação de que já existiam ferramentas prontas que já podiam fornecer esta funcionalidade. Portanto, a terceira abordagem foi a selecionada e dentre algumas ferramentas existentes optou-se pelo *Envers* que propõe um menor trabalho de configuração e de *refactoring* de código fonte, além de fazer parte do Hibernate que já era usado no projeto.

## Configuração do Hibernate Envers

A partir da versão 3.5 do Hibernate, o módulo do *Envers* foi incluído como um módulo core do Hibernate. Esta biblioteca é facilmente obtida no próprio site do Hibernate. No caso do projeto do SIExp, o servidor de aplicação JBoss AS 7.1.3 possui embutida a versão 4.2.17 do Hibernate como um de seus módulos, ou seja, o *Envers* já é habilitado no próprio servidor. Portanto, só houve o cuidado de garantir que as bibliotecas *hibernate-core, hibernate-entitymanager, hibernate-commons-annotations* e *hibernate-envers* estivessem referenciadas no ambiente de desenvolvimento para conseguirmos usar as anotações e classes do Envers em tempo de compilação da nossa aplicação.

Após este passo de habilitar as bibliotecas para a utilização do Hibernate na aplicação, foi realizado um estudo de todas as tabelas do banco de dados do SIExp para definir quais dados seriam auditados e quais não seriam. Esta foi uma ação importante, pois, pela simplicidade oferecida para a realização dos logs de auditoria parece tentador configurar toda e qualquer entidade como auditável. Porém, a seleção dos dados se torna fundamental a fim de evitar que uma quantidade enorme de dados desnecessários sejam registrados nos logs de auditoria podendo causar sobrecarga na infraestrutura computacional requerida para a aplicação. Os critérios de seleção foram baseados na sensibilidade e importância dos dados.

Uma vez que as entidades foram selecionadas e anotadas (exemplo de anotação já foi ilustrado na Figura 1 deste documento), partiu-se para uma definição de infraestrutura e arquitetura para os dados de log de auditoria. Optou-se por criar um esquema de banco de dados separado para as tabelas que armazenam os logs de auditoria com o intuito de facilitar as tarefas de administração das bases de dados tais como manter e restaurar backups, resolver problemas de desempenho etc.

O *Envers* disponibiliza algumas formas diferentes para a criação automática do modelo de tabelas de audito-

ria. Uma destas formas é usar a propriedade *hibernate.hbm2ddl.auto* que faz com que o próprio Hibernate crie todas as tabelas de auditoria necessárias, baseado nas configurações das entidades. Outra forma é gerar o esquema de forma programática usando a *classe org.hibernate.tool.EnversSchemaGenerator*. Por último, existe uma outra forma automática que é utilizar uma *task* da ferramenta *Ant*. No SIExp, a decisão foi não utilizar nenhuma destas formas automáticas e assim criar manualmente o esquema com as tabelas de auditoria seguindo o padrão previsto para o modelo de auditoria do *Envers* que está ilustrado na Figura 2.

No esquema de auditoria do SIExp havia a necessidade de armazenar a informação do responsável por cada revisão, portanto, incluímos a coluna usuario_id na tabela revisao_aud para este propósito. Além disso, implementamos a classe *SiexpRevEntity* que segue a mesma estrutura da classe exibida na Figura 3 e também criamos a classe *SiexpRevisionListener* que é análoga à classe ilustrada na Figura 4 com a devida implementação de obtenção do código do usuário utilizando os serviços específicos do SIExp.

O *Envers* possibilita que o usuário possa definir uma estratégia para a configuração de auditoria para as entidades de relacionamento, pois oferece algumas opções diferentes para a auditoria dos relacionamentos entre entidades. No SIExp, todos os atributos de entidades que representavam os relacionamentos do tipo *@OneToMany* do SIExp foram marcados com a anotação *@NotAudited* e as entidades alvo deste relacionamento foram marcadas com *@Audited*, fazendo com que as auditorias sempre acontecessem individualmente por tabela mesmo que a alteração ocorresse sempre em conjunto com uma outra entidade relacionada. Os relacionamentos do tipo *@ManyToOne* foram todos anotados com *@Audited(targetAuditMode=RelationTargetAuditMode.NOT_AUDITED)* e as entidades representantes de domínios (como status, tipos etc) que não possuem mudança em seus dados não foram selecionadas como entidades auditáveis.

A incorporação do *Envers* na aplicação do SIExp trouxe algum impacto para o processo de geração de mudanças no banco de dados, pois, como toda tabela no modelo original possui uma tabela “espelho” no modelo de auditoria, sempre que houver alguma alteração no modelo original (inclusão/remoção/alteração de tabelas e/ou colunas) esta mudança dever ser refletida no modelo de auditoria do SIExp, pois, se isso não acontecer e de acordo com o tipo de mudança o sistema deve lançar uma exceção do Hibernate no momento da auditoria ou alguma tabela nova pode não passar a ser auditada. Até que a equipe se familiarizasse com o novo processo ocorreram alguns casos de erros causados por uma falha na atualização do modelo de banco de dados de auditoria.

## 2.2. Testes de validação

Após as configurações, customizações e alterações realizadas, o SIExp ficou habilitado para fazer o registro de log de auditorias das entidades que foram selecionadas. Antes de disponibilizar a auditoria de entidades no ambiente de produção, foi planejada uma etapa de testes para verificar o funcionamento da ferramenta. Estes testes foram programados para que fossem realizadas as ações no sistema que permitissem executar todas as operações possíveis em todas as entidades (inclusão, modificação, remoção). Como já era esperado, alguns problemas foram encontrados nesta fase e serão detalhados nesta seção.

A primeira grande dificuldade encontrada foi a descoberta de que o Hibernate Envers não consegue realizar auditoria de operações efetuadas pela execução de comandos SQL nativos. Isto acontece porque o mecanismo de eventos do *Envers* só é acionado quando determinadas operações do *EntityManager* são executadas tais como *persist(), merge() e remove()*, não sendo acionado quando ocorre uma execução de um SQL nativo. Muitas operações de remoção em lote que são feitas no SIExp utilizam a estratégia de execução de SQL nativo para otimização de performance e, portanto, estas operações não estavam provocando o registro de mudanças no log de auditoria. Constatado este problema, foi realizada uma análise de todas as operações existentes que utilizam SQL nativo e para as quais a questão de performance não era crítica alteramos a implementação para a utilização do método *remove()* do *EntityManager*. Para as demais, tivemos que implementar um serviço próprio que insere os registros de auditoria. Este serviço é invocado pelos métodos de remoção de entidades que fazem uso de SQL nativo logo após a operação de exclusão. Este serviço é genérico para qualquer tipo de entidade, já que para as operações de remoção, o *Envers* grava todos os atributos da entidade como nulos.

Excetuando os casos de remoção em lote citados acima, tivemos dois casos para os quais tivemos que implementar serviços específicos de inserção de registros nas tabelas de auditoria, sendo que um deles era uma operação de inclusão em uma entidade que usava SQL nativo e a outra era um caso de uma entidade que possuía uma implementação de chave primária composta representada por uma outra classe para a qual o mecanismo de auditoria do *Envers* não funcionou.

Outro problema enfrentado foi devido a decisão inicial do projeto de se atribuir a chave primária das tabelas de auditoria como sendo sempre o id original da entidade acrescido do id da revisão. Foi detectado que em algumas transações do SIExp um mesmo registro de uma entidade poderia ser alterado mais do que uma vez e, nestes casos ocorria um erro de violação de chave-primária quando o *Envers* executava. Um exemplo deste problema surgiu em funcionalidades nas quais havia uma operação de modificação e outra de remoção na mesma transação, e portanto dois registros com a mesma chave eram enviados para a mesma tabela de auditoria. Para resolver estes casos de duplicação de chave primária na mesma transação, incluímos nas chaves de algumas tabelas o campo tipo_revisao, já que desta forma garantimos que numa mesma transação não poderia haver mais do que uma alteração do mesmo tipo.

Na tentativa de utilizar o Ant (THE APACHE SOFTWARE FOUNDATION, 2016) para gerar automaticamente o esquema de auditoria, a equipe esbarrou em um problema com uma biblioteca que não funcionava e o erro apontado pela exceção lançada não pôde ser detectado, por isso, esta foi uma das motivações para que se adotasse a geração manual do modelo de dados de auditoria.

Uma das formas de auditoria de banco de dados é registrar também as informações de quem e quando consultou as entidades do banco de dados. O *Envers* não se propõe a fazer este tipo de auditoria, portanto, caso se deseje realizar este tipo de log, é necessário a utilização de algum outro mecanismo.

## Conclusão

A utilização do *Envers* como uma solução para log de auditoria de entidade de banco de dados possui um excelente custo/benefício tendo em vista que sua configuração é rápida e simples. Embora a ferramenta possua algumas opções de customização, para caos nos quais os requisitos de auditoria forem extremamente exigentes e complexos, a adoção do Hibernate pode não ser ideal devido as limitações e dificuldades já citadas neste documento, como limitações de funcionamento com alguns tipos de implementação de chaves compostas, falta de suporte a log de consultas e a log de comandos SQL nativos. Entretanto, no caso do SIExp, a decisão se mostrou acertada já que o custo de sua implantação foi baixo comparado a outras opções e o benefício relacionado à segurança da informação já pôde ser sentido tanto em alguns incidentes ocorridos nos quais a existência do log de auditoria foi determinante para se encontrar a causa do problema quanto pela garantia de que todos os valores sensíveis incluídos, alterados e removidos estão sendo preservados.

## Referências

DataONE. **Data life cycle**. 2016. Disponível em: <https://www.dataone.org/data-life-cycle>. Acesso em: 7 nov. 2016.

HIBERNATE everythings data. Disponível em: <http://hibernate.org/>. Acesso em: 03 nov 2016.

HIBERNATE ENVERS. Disponível em: <http://hibernate.org/orm/envers/>. Acesso em: 03 nov 2016.

*Envers*. Disponível em: <http://docs.jboss.org/hibernate/orm/4.2/devguide/en-US/html/ch15.html>. Acesso em: 7 nov 2016.

BEN-NATAN, R. **Implementing database security and auditing**: a guide for DBAs, information security administrators and auditors. Burlington: Elsevier, 2005. 413 p.

YANG, L. Teaching database security and auditing. **ACM SIGCSE Bulletin**, v. 41, n 1, p. 241-245, Mar. 2009. Edição de Proceedings of the 40th ACM Technical Symposium on Computer Science Education, Chattanooga, 2009. Disponível em: <https://doi.org/10.1145/1539024.1508954>. Acesso em: 7 nov. 2016.

THE APACHE SOFTWARE FOUNDATION. **Apache Ant**. Disponível em: <http://ant.apache.org/>. Acesso em: 10 nov 2016.

**Comunicado Técnico, 124**

**Embrapa Informática Agropecuária**
Endereço: Caixa Postal 6041 - Barão Geraldo
13083-886 - Campinas, SP
Fone: (19) 3211-5700
www.embrapa.br/informatica-agropecuaria
www.embrapa.br/fale-conosco/sac/

MINISTÉRIO DA AGRICULTURA, PECUÁRIA E ABASTECIMENTO

**1ª edição publicação digital - 2016**





**Expediente**

**Supervisão editorial:** *Stanley Robson de Medeiros Oliveira, Suzilei Carneiro*

**Normalização bibliográfica:** *Maria Goretti Gurgel Praxedes*

**Revisão de texto:** *Adriana Farah Gonzalez*

**Editoração eletrônica:** *Suzilei Carneiro*

CGPE 13471