Gestão de Regras de Autorização Usando Modelo Conceitual

Alternative Title: Management of Authorization Rules Using Conceptual Model

Ricardo Diniz Sul,
Bruna Christina P. Brandão
Programa de Pós-Graduação em Informática (PPGI/UNIRIO)
Av. Pasteur, 458
Urca, Rio de Janeiro - RJ
+55 (21) 2542-6368
ricardo.diniz@uniriotec.br,
bruna.christina@uniriotec.br

Leonardo Guerreiro Azevedo
Programa de Pós-Graduação em Informática (PPGI/UNIRIO)
Av. Pasteur, 458
Urca, Rio de Janeiro - RJ
+55 (21) 2542-6368
IBM Research
Avenida Pasteur, 160
Botafogo, Rio de Janeiro – RJ
lga@br.ibm.com

Fernanda Baião,
Claudia Cappelli
Programa de Pós-Graduação em Informática (PPGI/UNIRIO)
Av. Pasteur, 458
Urca, Rio de Janeiro - RJ
+55 (21) 2542-6368
fernanda.baiao@uniriotec.br,
claudia.cappelli@uniriotec.br

RESUMO

Segurança da informação é um importante aspecto para construção de sistemas de informação. A gestão e execução das regras de autorização, que restringem a **quem** é permitido realizar uma **ação** na organização e sobre quais **informações,** são aspectos cruciais. Este trabalho apresenta uma ferramenta para gestão de regras de autorização de um framework para controle de acesso baseado em perfis de usuário. Através deste módulo, usuários do negócio são capazes de especificar regras de autorização utilizando um modelo baseado no metamodelo entidade-relacionamento. O módulo foi implementado empregando tecnologias de código aberto em um cenário de uma organização real que gerencia o acesso de vários sistemas de informação a uma mesma base de dados corporativa. Um exemplo de seu uso é apresentado, ilustrando a sua viabilidade e eficácia.

Palavras-chave

Regras de autorização; Gestão de regras de autorização; Metamodelagem

ABSTRACT

Information security is an important concern for information systems development. Managing and executing authorization rules (which constrain who is allowed to execute some action over which information) are crucial issues. This work presents a tool for managing authorization rules part of a role-based framework for access control. Business users may use this module to specify authorization rules using an ERM (entity-relationship model). The module was implemented using open-source technologies, in a real organization that is responsible for controlling the access of several information systems to a corporate database. An example of its use is presented, illustrating its viability and efficacy.

SBSI 2015, May 26–29, 2015, Goiânia, Goiás, Brazil.

Categories and Subject Descriptors

H.2.7 [Information Systems]: DATABASE MANAGEMENT Database Administration – *Security, integrity, and protection.*

General Terms

Security

Keywords

Authorization rules; Management of authorization Rules; Metamodeling

1. INTRODUÇÃO

Segurança da informação inclui aspectos de controle de acesso, autenticidade, auditoria, controle do fluxo de informação, disponibilidade e confidencialidade. Controle de acesso se apresenta como elemento central, sendo responsável em grande parte pela garantia das regras de autorização, um tipo específico de regras de negócio [3][9][14][16]. Regras de negócio têm como objetivo definir a estrutura de um negócio ou controlar ou influenciar o seu comportamento. Em particular, uma **regra de autorização** restringe a **quem** é permitido realizar uma **ação** e sobre quais **informações** esta ação pode ser exercida. Organizações tipicamente adotam (ou desenvolvem) sistemas para editar e executar (garantir) regras de autorização, ou replicam a sua implementação em seus sistemas legados.

Este trabalho apresenta uma proposta de modelagem conceitual e representação gráfica de regras de autorização em um nível de abstração conceitual, assim como o modelo lógico para armazenamento das regras em um banco de dados relacional e a geração automática da mesma para execução no *framework*. A modelagem conceitual representa a regra em um nível de abstração que elimina características técnicas de representação para o usuário do negócio.

O módulo proposto está no contexto do *framework* para gestão e execução de regras de autorização denominado (FARBAC (Flexible Approach for Role-Based Access Control) [1]. Este *framework* é composto por dois módulos: Gestão de Regras de Autorização (GRA) e Execução de Regras de Autorização (ERA). O módulo GRA é responsável pela criação, alteração,

visualização, composição, teste e simulação de regras de autorização, e o módulo ERA é responsável pela aplicação da regra de autorização em tempo de execução.

A representação da regra de autorização é baseada no modelo entidade-relacionamento (*ERM*). No ERM o mundo real é reduzido a entidades e relacionamentos entre essas entidades, onde cada entidade representa uma "coisa" ou um "conceito" que pode ser facilmente identificado [5]. O *ERM* permite representar um modelo cuja finalidade é descrever, de maneira mais próxima do usuário de negócio, os dados utilizados em um sistema ou pertencentes a um domínio.

O trabalho está organizado da seguinte forma. A Seção 2 apresenta a metodologia empregada para a execução deste trabalho. A Seção 3 apresenta o framework para regras de autorização. A Seção 4 apresenta como é realizada a representação e geração automática de regras de autorização. A Seção 5 apresenta a ferramenta implementada e um exemplo de uso da mesma. Finalmente, a Seção 6 apresenta a conclusão do trabalho e propostas de trabalhos futuros.

2. METODOLOGIA DO TRABALHO

Este trabalho foi realizado no contexto de construção de um framework para gestão e execução de regras de autorização para atender às necessidades de uma grande organização brasileira.

O framework foi dividido em dois módulos: gestão e execução de regras de autorização. Devido às prioridades da empresa, o foco inicial foi construir o módulo de execução de regras de autorização [1][12]. Em seguida, foi iniciado o projeto e desenvolvimento do módulo de gestão de regras de autorização descrito neste trabalho. A metodologia empregada para construir este módulo incluiu as seguintes etapas:

Etapa 1) Definir abordagem gráfica empregando modelagem conceitual de dados para especificação de regras de autorização que seja intuitiva para usuários de negócio. Definir cenários para análise do uso das abordagens criadas.

Etapa 2) Definir mecanismos para armazenamento e mapeamento das representações abstrata e concreta do banco de dados. A representação abstrata é utilizada para construir a interface que é apresentada para o usuário do negócio manipular para definir a regra de autorização. A representação concreta é utilizada para representar as informações necessárias para mapeamento da regra nos elementos físicos do banco de dados, as quais são utilizadas pelo módulo de execução de regras. Neste caso, foram escolhidos o modelo entidade-relacionamento (ER) para representação abstrata e o modelo relacional para a representação concreta. As regras apresentadas por [7] foram escolhidas para mapeamento do modelo ER para o modelo relacionado.

Etapa 3) Criar tabelas para armazenamento dos elementos dos modelos e mapeamentos: (i) Tabelas para armazenar os elementos do modelo ER; (ii) Tabelas para armazenar os elementos do modelo relacional; (iii) Tabelas para armazenar as regras utilizadas para mapeamento do modelo ER no modelo relacional.

Etapa 4) Elaborar consultas para recuperar informações dos conjuntos de tabelas (ER, relacional e de mapeamento) para construir as cláusulas da regra de autorização.

Etapa 5) Construir a ferramenta para permitir navegação no modelo conceitual definindo as entidades, relacionamentos, atributos e valores para os atributos, além do perfil e operação a serem considerados na regra. Foram definidas as bibliotecas para construção da ferramenta; a ferramenta foi construída e um exemplo foi utilizado para ilustrar sua viabilidade.

3. *FRAMEWORK* PARA REGRAS DE AUTORIZAÇÃO

FARBAC (Flexible Approach for Role-Based Access Control) é um *framework* composto dos módulos de gestão e execução de regras de autorização proposto por[1]. O aspecto da execução foi tratado anteriormente por Puntar *et al.* [12], que propôs o módulo ERA (Execução de Regras de Autorização) para execução em tempo real das regras de autorização sobre dados armazenados em bases de dados relacionais, utilizando tecnologias não intrusivas. O módulo ERA foi empregado como parte do mecanismo de propagação de identidade proposto por Leão *et al.* [8]. O presente trabalho trata o aspecto de gestão das regras de autorização e propõe a modelagem e implementação do módulo denominado GRA (Gestão de Regras de Autorização), que permite a especificação das regras de autorização por usuários de negócio e a geração automática das regras para serem persistidas em bancos de dados relacionais e garantidas pelos mecanismos de execução.

A arquitetura do *framework* proposto implementa os mecanismos de controle de acesso RBAC (*Role-Based Access Control*) e RDBAC (*Reflective Database Access Control*) na camada de dados dos sistemas de informação da organização, conforme proposto por [13]. No RBAC [6] define-se **quem** (quais perfis de usuários) pode realizar qual **ação** sobre qual **dado** ou funções. No mecanismo RDBAC, a regra de autorização é definida de forma reflexiva, baseando-se em dados já presentes no banco de dados [10]. Exemplos de regras que seguem o mecanismo RBAC (porque aluno e professor são papéis com responsabilidades e direitos definidos) e RDBAC (porque dados presentes no banco de dados são utilizados para aplicar a regra) são:

```
Um aluno deve ter acesso somente as suas próprias notas.        (R1)

Um professor deve ter acesso às notas de todos os alunos matriculados nas disciplinas que ele leciona.        (R2)
```

O módulo GRA (Gestão de Regras de Autorização) permite definir regras de acesso aos termos/conceitos da organização incluindo as operações que podem ser executadas por cada perfil existente. Todas as regras criadas neste módulo são armazenadas em um banco de dados de regras de autorização segundo um metamodelo proposto por Azevedo *et al.* [1]. O principal requisito deste módulo é que ele deve possuir uma interface que permita cadastrar conceitos e seus atributos, perfis, operações e relacionar estes elementos para formar uma regra de autorização. A Figura 1 ilustra a definição de uma regra de autorização no módulo GRA, na qual o conceito *Pedido* tem os atributos *identificador*, *data* e *valorTotal*. O perfil *Gerente local* pode *consultar* os pedidos cujo *valortotal* é menor do que determinado valor especificado pelo usuário, por exemplo, R$ 1.000,00 (um mil reais).

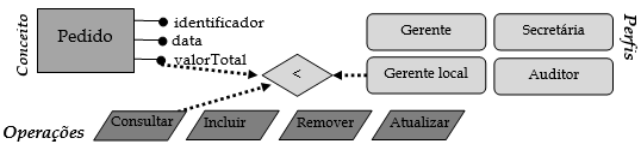

Figura 1 – Definição de regra de autorização na ferramenta de gestão (adaptado de [1])

O módulo ERA (Execução de Regras de Autorização) é responsável por garantir que as regras de autorização definidas previamente e armazenadas na base de regras sejam garantidas. Quando uma aplicação cliente envia uma consulta SQL para a base de dados corporativa, o módulo ERA intercepta a consulta, obtendo o comando SQL e a identificação do usuário que solicitou a execução da consulta. Em seguida, as regras de autorização são lidas do banco de regras de autorização. A consulta SQL então é automaticamente reescrita, em tempo real, a fim de garantir que o usuário só acesse os dados e execute a operação que ele tem direito. A consulta modificada é então enviada ao SGBD para ser executada e retornar os dados, que são repassados para a aplicação cliente.

4. DEFINIÇÃO E GERAÇÃO AUTOMÁTICA DE REGRA DE AUTORIZAÇÃO

Esta seção apresenta o passo-a-passo para definição e geração de regras de autorização, um exemplo de uso do módulo GRA, os metamodelos empregados e regras de mapeamento.

4.1 Definição da regra de autorização

Para a definição das regras de autorização pelo usuário do negócio, o GRA adotou a representação gráfica de um modelo conceitual de dados. Dentre os diversos modelos e notações disponíveis na literatura (MER – Modelo Entidade-Relacionamento [4]), UML [11] e ontologias [2], optou-se por adotar o MER, por ser o de notação mais simples, e de mais fácil entendimento. A representação e geração automática de uma regra seguem os seguintes passos:

1. Definir perfil: Usuário escolhe o perfil ao qual a regra deve ser aplicada.
2. Definir regra: Usuário escolhe a entidade para a qual a regra deve ser aplicada, navega pelo diagrama ER selecionando os relacionamentos, entidades relacionadas e seus atributos que fazem parte da definição da regra, opcionalmente definindo valores específicos para tais atributos, que devem ser restritos pela regra.
3. Gerar regra: Usuário solicita geração da regra, e o sistema automaticamente captura os dados armazenados durante a definição da regra, aplica as regras de mapeamento necessárias e gera a regra de autorização na base de regras.

Para exemplificar a proposta deste trabalho foi escolhido o domínio do benchmark TPC-H [15], o qual é uma especificação de benchmark com uma ampla relevância na indústria. Ele simula o cenário de uma aplicação de apoio à decisão. A **Erro! Fonte de referência não encontrada.** apresenta o modelo ER do TPC-H. Para ilustrar a proposta, considere a regra de autorização R3.

Em R3, o perfil a ser considerado é "gerente da América do Norte e Ásia". A entidade à qual a regra deve ser aplicada é "Pedido" e a operação é "consultar". A regra propriamente dita envolve os "pedidos de clientes localizados no hemisfério norte e que se encontram nas regiões da Ásia ou América". Para cadastrar esta regra, o usuário deve escolher os seguintes elementos do MER: `Order` (entidade na qual a regra deve ser aplicada); `Customer`, `Nation` e `Region` (outras entidades para especificação da regra); `cus_ord`, `cus_nat`, `nat_reg` (relacionamentos); `Nation.hemisphere`, `Region.name` (atributos); `Nation.hemisphere = "N"`; `Region.name = "Asia or America"`(valores para os atributos).

`Um gerente da América do Norte e Ásia deve somente consultar pedidos de clientes localizados no hemisfério norte e que se encontram nas regiões da Ásia ou América.` (R3)

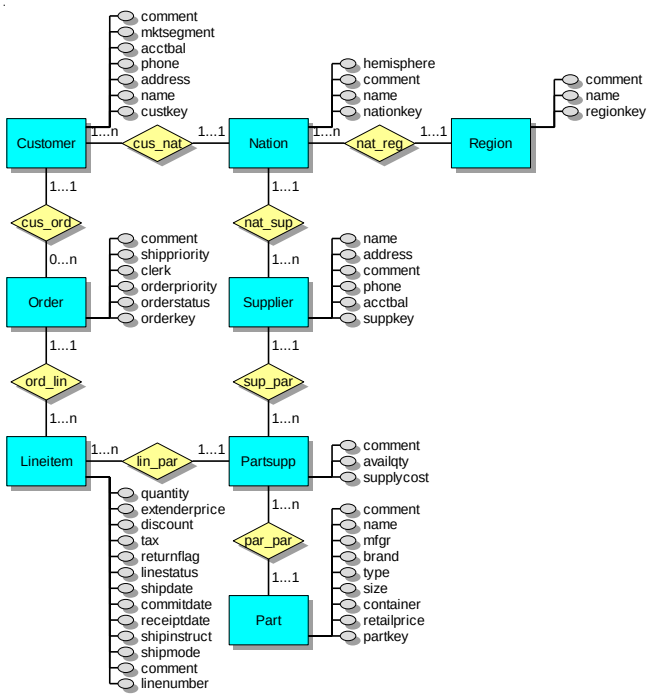

Figura 2 - Modelo ER do TPC-H

4.2 Mapeamento entre metamodelos

Para a exibição e mapeamento das entidades a serem consideradas na descrição da regra bem como a geração da regra considerando os objetos existentes fisicamente no banco de dados é necessário que estas informações sejam armazenadas em modelos mais abstratos. Ou seja, é necessário que os esquemas conceitual e lógico do banco de dados sejam eles próprios armazenados em modelos. Em outras palavras, é necessário se ter metamodelos para armazenar os objetos presentes nos modelos de cada domínio tratado pela organização. Um metamodelo é um modelo que guia a especificação de outros modelos, capturando propriedades e características essenciais em um nível de abstração mais elevado. Isto inclui, no caso do metamodelo ER, por exemplo, as entidades, seus relacionamentos e atributos. A Figura 3 e a Figura 4 apresentam os metamodelos do modelo conceitual ER e do modelo lógico relacional (do banco de dados).

Existem diferentes regras de mapeamento de modelo conceitual ER para modelo lógico relacional. Neste trabalho são empregadas as regras apresentadas por Heuser [7].

Para tradução das entidades os passos são: (i) Cada entidade para uma tabela; (ii) Cada atributo como uma coluna da tabela; (iii) Atributos identificadores para chave primária da tabela. Por exemplo, uma entidade *Pessoa* com atributos *código (atributo identificador)*, *nome* e *endereço* é mapeada para a tabela *Pessoa* com colunas *codigoPess (chave-primária)*, *nome* e *endereço*.

Para tradução dos relacionamentos existem três alternativas, dependendo da cardinalidade (máxima e mínima) do

relacionamento: (i) Tabela própria: o relacionamento é implementado através de uma tabela própria que contém as colunas correspondentes aos identificadores das entidades relacionadas e colunas correspondentes aos atributos do relacionamento; (ii) Adição de colunas: na tabela correspondente à entidade com cardinalidade máxima 1, adicionam-se colunas correspondentes ao identificador da entidade relacionada, onde estas colunas formam uma chave estrangeira em relação à tabela que implementa a entidade relacionada e as colunas correspondentes aos atributos do relacionamento; (iii) Fusão de tabelas: cria-se uma única tabela para todos os atributos das entidades relacionadas, bem como os atributos eventualmente existentes no relacionamento entre elas.

Considerando as melhores alternativas para relacionamentos temos as seguintes soluções, onde "_" indica 0 ou 1.

- Relacionamentos (0,1) × (_,1): adição de colunas.
- Relacionamentos (1,1) × (1,1): fusão de tabelas.
- Relacionamentos (_,1) × (_, n): adição de colunas.
- Relacionamentos (_, n) × (_, n): tabela própria.

Para relacionamentos com grau maior que dois em geral, o relacionamento é transformado em uma tabela.

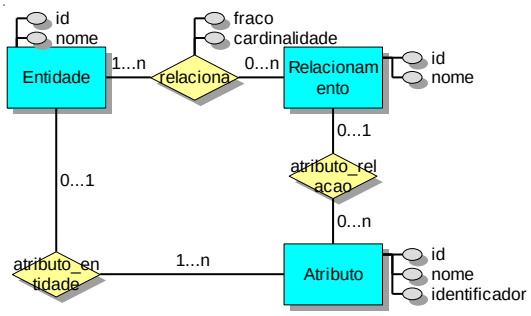

Figura 3 - Metamodelo do Modelo conceitual ER

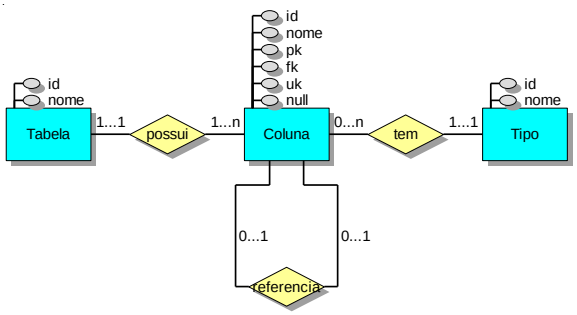

Figura 4 - Metamodelo do Modelo lógico relacional

4.3 Geração automática de regras de autorização

O usuário seleciona as entidades, relacionamentos e os atributos no qual a regra de autorização será aplicada, define valores de atributos e parâmetros de execução, se for o caso, e finalmente requisita sua geração. Dependendo dos elementos selecionados pelo usuário, a regra de autorização a ser gerada é de um dos possíveis tipos: (i) "Existe relacionamento": O usuário seleciona a entidade sobre a qual a regra deve ser aplicada e um relacionamento desta entidade com outra entidade. Ou seja, o usuário define uma regra de autorização restringindo o acesso de um usuário aos registros da entidade que atendem o caminho definido no relacionamento (por exemplo, regra R3); (ii) "Própria entidade": O usuário define a entidade sobre a qual a regra de autorização deve ser aplicada e valores de atributos apenas desta entidade; (iii) "Predicado dinâmico": a regra de autorização definida deve obter informações em tempo de execução (por exemplo, o *login* do usuário).

A geração automática da regra de autorização realiza as seguintes atividades, dependendo do tipo da regra. Para o tipo "Existe relacionamento", a regra é montada considerando a melhor alternativa de tradução de relacionamento. Os operadores SQL a serem utilizados para esta regra são `in` ou `exists`, pois busca-se verificar se o relacionamento existe. Considerando os três tipos de relacionamentos, temos:

i. Existe relacionamento 1 × N: Este relacionamento é resolvido por adição de coluna. Dessa forma, as colunas que fazem parte da chave primária (chamemos *c1*, *c2* ... *cn*) da tabela do lado N (chamemos de *TN*) são incluídas na tabela do lado 1 (chamemos de *T1*). Por simplificação, consideraremos que as colunas foram migradas com o mesmo nome. Estas colunas serão utilizadas para montar o predicado com `in` ou `exists`. O operador `in` é gerado se apenas uma coluna foi adicionada na tabela do lado 1, resultando em:

```
T1.c1 in (SELECT TN.C1 FROM TN)
```

O operador `exists` pode ser usado para o caso de uma ou mais colunas adicionadas na tabela do lado 1. Ele corresponde à junção entre as tabelas participantes do relacionamento, da seguinte forma:

```
exists (SELECT 1 FROM TN WHERE T1.c1 =
TN.c1 AND T1.c2 = TN.c2 ... AND T1.cn =
TN.cn)
```

Caso outros relacionamentos precisem ser resolvidos, acrescentam-se tabelas na cláusula FROM da consulta e predicados de junção na cláusula WHERE de acordo com o tipo de relacionamento e a tradução utilizada para o relacionamento. Os valores de atributos entram como cláusulas de seleção no WHERE, por exemplo, `AND TK.ck < 1000`.

ii. "Própria entidade": Neste caso, é necessário identificar como a entidade e os atributos foram mapeados para o modelo lógico e gerar cláusulas do tipo `<tabela para qual a entidade foi mapeada>.<coluna para a qual a entidade foi mapeada><operador><valor>`. Por exemplo, considerando que a entidade `E` foi a entidade escolhida e ela foi mapeada para a tabela `T`, que os atributos `a1`, `a2` e `a3` foram escolhidos e mapeados para as colunas `c1`, `c2` e `c3`, que o operador escolhido foi "=", e que os valores `v1`, `v2` e `v3` foram definidos para cada atributo, o predicado gerado teria a forma:

$$T_1.c_1 = v_1 \text{ AND } T_2.c_2 = v_2 \text{ AND } T_3.c_3 = v_3$$

iii. "Predicado dinâmico": Neste caso, o valor do predicado é atribuído em tempo de execução. Um exemplo de cláusula seria `TK.ck = $User`, onde `TK` e `ck` são a tabela e a coluna correspondente ao mapeamento da entidade e atributo e `$User` indica o *login* do usuário, obtido em tempo de execução.

5. A FERRAMENTA CSAR

A ferramenta CSAR (Conceptual Specification of Authorization Rules) foi desenvolvida para permitir o cadastro de regras de autorização considerando a proposta de geração automática. A

interface do CSAR é apresentada na Figura 5. O "Painel de grafo do domínio" (Figura 5.a) apresenta o modelo conceitual construído utilizando a biblioteca "prefuse" (http://prefuse.org/). Prefuse é um conjunto de ferramentas de software para criar visualizações interativas ricas de dados. O *toolkit* Prefuse original fornece um quadro de visualização para a linguagem de programação Java (http://code.google.com/p/gra/). O "Painel de edição de regras de autorização" (Figura 5.b) exibe informações sobre entidades, relacionamentos e atributos selecionados no modelo e permite definir valores para os atributos escolhidos. Na parte inferior direita da figura, há um botão que ao ser clicado executa o algoritmo de geração de predicado para ser armazenado na base de dados do framework.

Como exemplo de uso da aplicação, considere a regra R3. Para este caso, o usuário irá escolher a entidade `Pedido` sobre a qual a regra será aplicada. Em seguida, selecionar o relacionamento `Compra` e `Cliente` (visto que a regra restringe o acesso a pedidos de cliente). Considerando que o relacionamento compra é 1 × N, a regra é do tipo "Existe relacionamento 1 × N". Para este caso, há duas opções de geração, `in` e `exists`.

Para regras via relacionamento com `in` e que o relacionamento foi resolvido com adição de coluna, os elementos a serem considerados são a entidade na qual a regra será aplicada e a chave estrangeira herdada da segunda entidade selecionada pelo usuário. Neste caso, a entidade `Pedido (order)` recebe `codigoDeCliente (custkey)` de `Cliente (customer)` resultando na cláusula: `order.custkey in (select customer.custkey from customer)`.

Para a cláusula `from` é necessário considerar o mapeamento das outras entidades selecionadas. Logo as cláusulas que formarão o `from` deverão incluir `customer`, `nation` e `region`, uma vez que a regra trata de `clientes (customer)` de `nações (nation)` do hemisfério norte e que se encontram na `região (region)` da Ásia ou América.

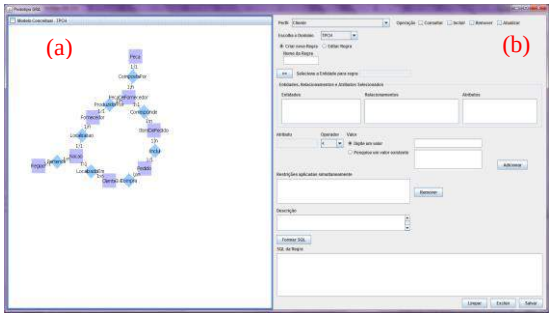

Figura 5 - CSAR (Conceptual Specification of Authorization Rules tool)

Em seguida, considera-se como foram mapeados os `joins`, neste caso `Compra`, `LocalizadoEm` e `Pertence`.

O relacionamento `LocalizadoEm` (1 × N) foi resolvido com a adição da coluna `codigoDeNacao (nationkey)` em `Cliente(customer)`. Logo, se traduz em: `customer.nationkey=nation.nationkey`.

O relacionamento `Pertence` (1 × N) foi resolvido com a adição da coluna `codigoDeRegiao (regionkey)` em `Nacao (nation)`. Este se traduz em `nation.regionkey=region.regionkey`.

Consideram-se, em seguida, os valores para os atributos utilizados para seleção sobre `Nacao (nation)` (apenas as nações do hemisfério norte, ou seja, "`nation.hemisphere='n'`") e `Regiao (region)` (apenas as regiões Ásia ou América, ou seja, "`region.name in ('Asia','America')`"). Estas restrições são combinadas com o conectivo lógico `AND`, resultando em:

```
nation.hemisphere='n' AND region.name in
('Asia','America').
```

No final da geração, chega-se ao predicado da Figura 6.

```
order.custkey in (
select customer.custkey
from customer, nation, region
where     customer.nationkey=nation.nationkey
AND  nation.regionkey=region.regionkey AND
      nation.hemisphere='n' AND
       region.name in ('Asia','America'))
```

Figura 6 – Predicado para a regra R3

5.1 Implementação da geração automática de regras de autorização

Esta seção descreve o passo a passo para a geração automática de regras de autorização considerando tabelas de mapeamento entre os metamodelos conceitual e lógico, que explicita quais elementos do modelo conceitual foram mapeados em elementos do modelo lógico, assim como exemplos de aplicação da regra.

Um elemento do metamodelo conceitual pode ser mapeado em vários elementos do metamodelo lógico, e um elemento do metamodelo lógico pode ser mapeado em vários elementos do metamodelo conceitual, sendo o mapeamento realizado como o apresentado na Figura 7.

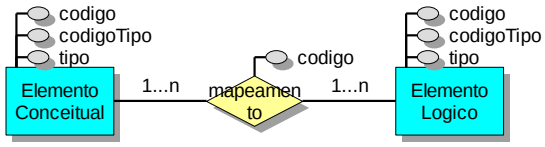

Figura 7 –Mapeamento entre elementos do modelo conceitual e elementos do modelo lógico

A partir deste diagrama, foi gerado um banco de dados correspondente com três tabelas, descritas a seguir.

A tabela `CONCEITUAL` armazena informações sobre os elementos do modelo conceitual, e possui as colunas: `CNT_ID`: chave-primária; `CNT_CON_ID:` identificador do elemento do modelo conceitual (por exemplo, `Order`, `custKey`); `CNT_TIPO`: tipo do elemento conceitual (por exemplo, `entidade`, `relacionamento` ou `atributo`).

A tabela `LOGICO` armazena informações sobre o modelo lógico, com as colunas: `LGC_ID`: chave-primária; `LGC_LOG_ID:` identificador do elemento do modelo lógico (por exemplo, `CUSTOMER` ou `CUS_ORD`); `LGC_TIPO`: tipo do elemento do modelo lógico (por exemplo, `TABELA` ou `COLUNA`).

Por último, temos a tabela `MAPEAMENTO` que armazena como um elemento do modelo conceitual foi mapeado em um elemento do modelo lógico. As colunas são: `MAP_ID`: chave-primária; `MAP_CNT_ID`: referencia o registro do elemento conceitual na tabela `CONCEITUAL`; `MAP_LGC_ID`: referencia o registro do elemento lógico na tabela `LOGICO`.

Para obter os mapeamentos realizados de entidades, relacionamentos e atributos, é necessário realizar consultas de nas tabelas do conjunto de mapeamentos, ou seja, de acordo com a solução adotada de mapeamento conceitual para lógico, uma consulta deve ser feita nas tabelas que armazenam as informações do metamodelo. Por exemplo, para o caso de relacionamento 1 × N via adição de coluna, a consulta para descobrir o código da coluna que mapeou o relacionamento é a seguinte:

```
SELECT COL.COL_ID
FROM TABELA TAB INNER JOIN COLUNA COL
ON TAB.TAB_ID=COL.COL_TAB_ID
WHERE TAB.TAB_NOME = '<NOME DA TABELA>'
    AND COL.COL_NOME='<NOME DA COLUNA>'
```

Como outro exemplo, para o mapeamento dos atributos é necessário realizar consultas de acordo com a estrutura do banco de dados para relacionar o código de um elemento da tabela `ATRIBUTO` do metamodelo conceitual com o código de outro elemento da tabela `COLUNA` do metamodelo lógico. Consultas foram omitidas devido à restrição de espaço.

A entrada do algoritmo para geração automática da regra inclui: seleção da entidade sobre a qual a regra será aplicada e seleção das entidades, relacionamentos e atributos envolvidos na regra. Para o tipo de regra "Existe relacionamento", a regra pode ser montada empregando os operadores `in` ou `exists`, pois busca-se verificar se o relacionamento existe. Para descobrir como o relacionamento foi mapeado, pode-se usar a consulta apresentada Figura 8.

i. O operador `in` só pode ser gerado se o relacionamento foi resolvido com adição de apenas uma coluna. O nome da coluna e da tabela referenciada é obtido executando a consulta apresentada na Figura 9, que, neste caso, tem que retornar apenas um registro. Seja *tabmap* a tabela onde a regra será aplicada, *colmap* a coluna correspondente ao mapeamento do relacionamento para coluna, *tabref* a tabela referenciada e *colref* a coluna correspondente, o predicado é construído como apresentado na Figura 10.

ii. O operador `exists` pode ser usado para o caso de uma ou mais colunas adicionadas na tabela do lado 1. A ideia é a mesma que a usada para `in`, mas substituindo o `in` por `exists`. Logo, a consulta apresentada na Figura 9 pode ser utilizada para obter as informações de mapeamento e o predicado gerado segue formato apresentado na Figura 11.

```
SELECT REL_NOME, L.LGC_TIPO
FROM CONCEITUAL.RELACIONAMENTO R
INNER JOIN CONCEITUAL CON
     ON ( (CON.CNT_CON_ID= R.REL_ID) AND
     (CON.CNT_TIPO='relacionamento') )
INNER JOIN MAPEAMENTO M
     ON (M.MAP_CNT_ID = CON.CNT_ID)
INNER JOIN LOGICO L
     ON L.LGC_ID=M.MAP_LGC_ID
WHERE REL_NOME='<NOME DO RELACIONAMENTO>'
```

Figura 8 - Consulta para obter mapeamento realizado do conceitual-lógico na tabela de mapeamento

```
SELECT  R.REL_NOME  RELACIONAMENTO,TB.TAB_NOME
TABMAP, C.COL_NOME COLMAP,
     T.TAB_NOME TABREF, CREF.COL_NOME COLREF
FROM CONCEITUAL.RELACIONAMENTO R
INNER JOIN CONCEITUAL CNT
     ON R.REL_ID=CNT.CNT_CON_ID
INNER JOIN MAPEAMENTO M
     ON (M.MAP_CNT_ID=CNT.CNT_ID AND
     CNT.CNT_TIPO = 'relacionamento')
  INNER JOIN LOGICO LGT
       ON LGT.LGC_ID=M.MAP_LGC_ID
  INNER JOIN COLUNA C
       ON C.COL_ID = LGT.LGC_LOG_ID
  INNER JOIN COLUNA CREF
       ON C.COL_REF_ID = CREF.COL_ID
  INNER JOIN TABELA T
       ON T.TAB_ID = CREF.COL_TAB_ID
  INNER JOIN TABELA TB
       ON C.COL_TAB_ID = TB.TAB_ID
  WHERE REL_NOME='<NOME DO RELACIONAMENTO>'
```

Figura 9 - Consulta para obter nome da coluna utilizada no conceitual-lógico na tabela de mapeamento

```
TABMAP.COLMAP in (SELECT TABREF.COLREF FROM
TABREF)
```

Figura 10 - Predicado gerado para in com tabela de mapeamento

```
EXISTS (SELECT 1 FROM TABREF WHERE TABMAP.COLMAP =
TABREF.COLREF)
```

Figura 11 - Predicado gerado para exists com tabela de mapeamento

A proposta apresentada considera o primeiro relacionamento selecionado a partir da entidade sobre a qual deseja-se aplicar a regra. No entanto outros relacionamentos podem ter sido escolhidos e valores podem ter sido definidos para os atributos. Dessa forma, outros relacionamentos precisam ser resolvidos. Isto é feito acrescentando-se tabelas na cláusula `FROM` da consulta de acordo com o mapeamento das entidades. A consulta apresentada na Figura 12 é utilizada para descobrir como a entidade foi mapeada e a consulta apresentada na Figura 13 é utilizada para descobrir o nome da tabela no caso da entidade ter sido mapeada para tabela.

```
SELECT CON.CNT_TIPO,ENT_NOME, L.LGC_TIPO
FROM CONCEITUAL.ENTIDADE E
INNER JOIN CONCEITUAL CON
     ON ( (CON.CNT_CON_ID= E.ENT_ID) AND
          (CON.CNT_TIPO='entidade') )
INNER JOIN MAPEAMENTO M
     ON (M.MAP_CNT_ID = CON.CNT_ID)
INNER JOIN LOGICO L ON L.LGC_ID=M.MAP_LGC_ID
WHERE ENT_NOME='<NOME DA ENTIDADE>'
```

Figura 12 - Consulta para obter mapeamento de entidade <entidade>

```
SELECT CON.CNT_TIPO, ENT_NOME,
       L.LGC_TIPO, T.TAB_NOME
FROM CONCEITUAL.ENTIDADE E
INNER JOIN CONCEITUAL CON
     ON ( (CON.CNT_CON_ID= E.ENT_ID) AND
        (CON.CNT_TIPO='entidade') )
INNER JOIN MAPEAMENTO M
     ON (M.MAP_CNT_ID = CON.CNT_ID)
INNER JOIN LOGICO L ON L.LGC_ID=M.MAP_LGC_ID
INNER JOIN TABELA T ON T.TAB_ID = L.LGC_LOG_ID
WHERE ENT_NOME='<NOME DA ENTIDADE>'
```

Figura 13 - Consulta para obter nome da tabela no caso de mapeamento de entidade para tabela

Além disso, deve-se identificar os predicados a serem acrescentados na cláusula WHERE de acordo com os atributos escolhidos pelo usuário. Um exemplo é a escolha de atributo *atr* de uma entidade *ent*. O mapeamento destes atributos entra como cláusulas de seleção no WHERE, por exemplo, `AND TK.ck < 1000`, considerando a necessidade de selecionar, no caminho de navegação, os registros da tabela `TK` que tenham a coluna `ck` com valores menor do que `1000`. Neste caso, é necessário, por exemplo, identificar como foi feito o mapeamento da entidade (usando as consultas apresentadas na Figura 12 e Figura 13) e

mapeamento do atributo. A consulta apresentada na Figura 14 identifica como o atributo foi resolvido. A consulta apresentada na Figura 15 pode ser utilizada para obter o nome da coluna e da tabela correspondente ao mapeamento do atributo.

```
SELECT CON.CNT_TIPO, E.ENT_NOME,
       ATR_NOME, L.LGC_TIPO
FROM CONCEITUAL.ATRIBUTO ATR
INNER JOIN CONCEITUAL CON
       ON ( (CON.CNT_CON_ID= ATR.ATR_ID) AND
            (CON.CNT_TIPO='atributo') )
INNER JOIN CONCEITUAL.ENTIDADE E
       ON E.ENT_ID = ATR.ATR_ENT_ID
INNER JOIN MAPEAMENTO M
       ON (M.MAP_CNT_ID = CON.CNT_ID)
INNER JOIN LOGICO L ON L.LGC_ID=M.MAP_LGC_ID
WHERE ATR_NOME='<NOME DO ATRIBUTO>'
AND E.ENT_NOME='<NOME DA ENTIDADE>'
```

Figura 14 - Consulta para identificar mapeamento de entidade <atributo>

```
SELECT CON.CNT_TIPO, ATR_NOME,
       L.LGC_TIPO, COL_NOME, TAB_NOME
FROM CONCEITUAL.ATRIBUTO ATR
INNER JOIN CONCEITUAL CON
       ON ( (CON.CNT_CON_ID= ATR.ATR_ID) AND
            (CON.CNT_TIPO='atributo') )
INNER JOIN CONCEITUAL.ENTIDADE E
       ON E.ENT_ID = ATR.ATR_ENT_ID
INNER JOIN MAPEAMENTO M
       ON (M.MAP_CNT_ID = CON.CNT_ID)
INNER JOIN LOGICO L ON L.LGC_ID=M.MAP_LGC_ID
INNER JOIN COLUNA COL ON COL.COL_ID = L.LGC_LOG_ID
INNER JOIN TABELA T ON T.TAB_ID=COL.COL_TAB_ID
WHERE ATR_NOME='<NOME DO ATRIBUTO>'
AND E.ENT_NOME='<NOME DA ENTIDADE>'
```

Figura 15 - Consulta para obter nome da coluna e da tabela correspondente ao mapeamento do atributo

Em regras aplicadas sobre a própria entidade, deve-se identificar a qual tabela a entidade foi mapeada e a qual coluna o atributo foi mapeado com a consulta da Figura 16.

```
SELECT COL_NOME COLMAP, TAB_NOME TABMAP
FROM CONCEITUAL.ATRIBUTO ATR
INNER JOIN CONCEITUAL CON
       ON ( (CON.CNT_CON_ID= ATR.ATR_ID) AND
            (CON.CNT_TIPO='atributo') )
INNER JOIN CONCEITUAL.ENTIDADE E
       ON E.ENT_ID = ATR.ATR_ENT_ID
INNER JOIN MAPEAMENTO M
       ON (M.MAP_CNT_ID = CON.CNT_ID)
INNER JOIN LOGICO L ON L.LGC_ID=M.MAP_LGC_ID
INNER JOIN COLUNA COL ON COL.COL_ID = L.LGC_LOG_ID
INNER JOIN TABELA T ON T.TAB_ID=COL.COL_TAB_ID
WHERE ATR_NOME='<NOME DO ATRIBUTO>'
AND E.ENT_NOME='<NOME DA ENTIDADE>'
```

Figura 16 - Consulta para obter o nome da coluna e da tabela correspondente no metamodelo lógico

A partir da tabela e da coluna mapeada é possível montar a regra através de predicado com `IN` ou `EXISTS`. Predicado do tipo `IN` é apresentado na Figura 17 e do tipo `EXISTS` na Figura 18.

```
COLMAP IN (SELECT COLMAP FROM TABMAP)
```

Figura 17 - Formato do predicado que deve ser gerado para in com tabela de mapeamento

```
EXISTS (SELECT 1 FROM TABMAP)
```

Figura 18 - Formato do predicado que deve ser gerado para exists com tabela de mapeamento

5.2 Exemplo

Para geração da regra de autorização R3 (“Um gerente da América do Norte e Ásia deve somente acessar pedidos de clientes localizados no hemisfério norte e que se encontram nas regiões da Ásia ou América”), o usuário de negócio utiliza a interface gráfica da ferramenta CSAR para escolher as entidades, relacionamentos e atributos que devem fazer parte da regra, como apresentado na Figura 19. Os elementos não selecionados aparecem sombreados, ou com o tom de cor mais claro – no qual os elementos selecionados ficam com a cor vermelha. A Figura 19 apresenta o caminho percorrido pelo usuário para criar a regra R3, com inicio na entidade *Pedido*, em seguida escolhendo o relacionamento *compra*, entidade *Cliente*, relacionamento *LocalizadoEm*, entidade *Nacao* com duplo click para exibir seus atributos selecionando *hemisferio*, o relacionamento *Pertence*, a entidade *Regiao* e o seu atributo *nome*.

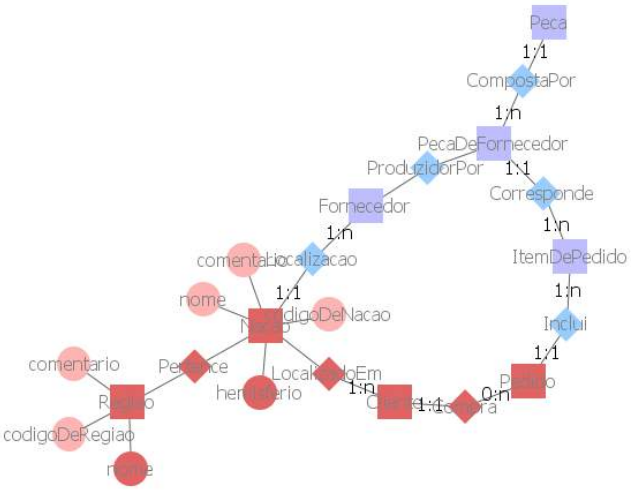

Figura 19 - Entidades, relacionamentos e atributos selecionados no modelo conceitual

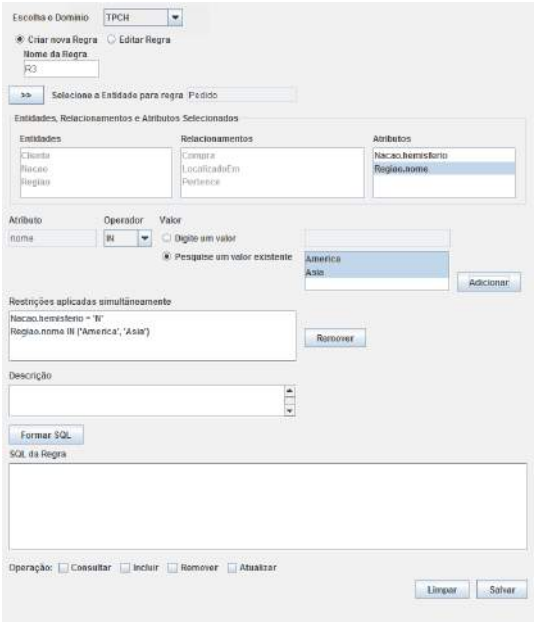

Figura 20 - Definição de valores para atributos para montar os predicados das regras

Após a escolha das informações no grafo, o usuário solicita que os elementos escolhidos sejam apresentados nos campos do lado direito da interface da ferramenta a fim de que ele possa explicitar as demais informações para a regra. Em seguida, o usuário informa os valores dos atributos para concluir a montagem da regra. A **Erro! Fonte de referência não encontrada.** apresenta os valores escolhidos e a especificação de valores para os atributos nome da região e hemisfério da nação a fim de definir os predicados das regras. Ao solicitar a geração da regra, é gerado o predicado apresentado na Figura 6.

Para gerar o predicado, o sistema identifica que o tipo é "Existe relacionamento". O sistema usa as tabelas de mapeamento do conceitual para o lógico. No modelo conceitual, estes elementos são as entidades `Pedido`, `Cliente`, `Nacao`, `Regiao`, os relacionamentos `Compra`, `LocalizadoEm`, `Pertence`, e os Atributos `hemisferio` de `nacao` e `nome` de `regiao`. No modelo lógico, os elementos são as tabelas `order`, `customer`, `nation`, `region`, as colunas que mapeiam relacionamentos `custkey` de `order`, `nationkey` de `customer` e `regionkey` de `nation`, as colunas que mapeiam atributos `hemisphere` de `nation` e `name` de `region`. As consultas apresentadas na Seção 5.1 são utilizadas para obter estes mapeamentos e montar o predicado.

6. CONCLUSÃO

Este trabalho propõe o GRA, um módulo que dá suporte à gerência de regras de autorização como parte importante da implementação do conceito de segurança da informação em sistemas de informação organizacionais.

A principal contribuição do módulo proposto é estender o estado da arte no suporte à definição e à geração automática de regras de autorização, além de implementar os mecanismos de controle de acesso RBAC e RDBC de forma não intrusiva. A definição das regras de autorização faz uso de representações em um nível de abstração alto, podendo por isso ser facilmente realizada por usuários de negócio. O modelo ER foi adotado na proposta pela simplicidade de seus construtos, mas outros modelos conceituais podem ser adotados, de forma complementar, explorando suas extensões e maior poder de expressividade. As regras de autorização definidas são armazenadas em uma base de dados e, a partir de um conjunto de regras flexíveis e parametrizadas, são traduzidas seguindo uma estratégia baseada no mapeamento entre os metamodelos conceitual ER (que representa a definição da regra) e lógico (que representa a implementação da regra sobre um SGBD relacional). A regra de autorização gerada é então armazenada em um repositório de regras, que pode ser acessado por mecanismos de Execução de Regras de Autorização para garantir o cumprimento das regras em tempo de execução, restringindo o acesso de quaisquer aplicações a informações não autorizadas pelas regras de autorização geradas.

O módulo GRA foi implementado em uma ferramenta gráfica CSAR, compreendendo a elaboração de metamodelos para o modelo conceitual ER e para o modelo lógico relacional, além de um metamodelo genérico de mapeamento entre eles, tornando seu uso flexível e parametrizável para qualquer domínio de conhecimento. A ferramenta foi utilizada em uma organização real, utilizando o cenário do TPC-H como prova de conceito.

Como trabalhos futuros pretende-se estender a ferramenta para tratar regras que envolvam hierarquia, caminhos distintos para a mesma entidade, e auto relacionamentos.

7. REFERÊNCIAS

[1] Azevedo, L.G., Puntar, S., Thiago, R., Baião, F., Cappelli, C., 2010. A Flexible Framework for Applying Data Access Authorization Business Rules. In *Proceedings of the 12th International Conference on Enterprise Information Systems* (ICEIS 2010), pp. 275-280.

[2] Baader, F., Calvanese, D., Mcguinness, D. L., Nardi, D., Atelschneider, P. F. (2003) *The Description Logic Handbook: Theory, Implementation, Applications*. Cambridge University Press, Cambridge, UK, 2003.

[3] Calì, A., Martinenghi, D (2008). Querying data under access limitations. In *Proceedings of IEEE 24th International Conference on Data Engineering*, pp. 50 – 59, 2008.

[4] Chen, P (1976). The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems,*Vol. 1 Issue 1, p. 9-36, USA.

[5] Choobineh, J. (1997) A meta ERM and its relational translation for a database design system. In *Proceedings of the 30th Hawaii International Conference on System Sciences: Advanced Technology*, vol 5., Washington, DC.

[6] Ferraiolo, D.F. e Khun, D. R. (1992) Role-Based Access Control. 15th National Computer Security Conference, pp. 554-563, Baltimore, Maryland, USA.

[7] Heuser, C.A. (2009) *Projeto de banco de dados*. Bookman.

[8] Leao, F., Azevedo, l. G., Santana, T, Baião, F., Cappelli, C. (2014) Controle de Acesso a Dados com Propagação de Identidade em Aplicações Web baseadas em Serviços. iSys: Revista Brasileira de Sistemas de Informação, v. 7, p. 48-65.

[9] Murthy, R., Sedlar, E (2007). Flexible and efficient access control in Oracle. *ACM SIGMOD* 2007, pp. 973-980,

[10] Olson, L. E.; Gunter, C. A.; E. Madhusudan, P. (2008) A formal framework for reflective database access control policies. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, pp. 289-298.

[11] OMG (2007) OMG Unified Modeling Language (OMG UML), Infrastructure, v. 2.1.2, 2007. Disponível em: <http://www.omg.org/spec/UML/2.1.2/>.

[12] Puntar, S., Azevedo, L., Baião, F., Cappelli, C. (2011) Implementing Flexible and Efficient Authorization Business Rules in Information Systems. In *Proceedings of IADIS Applied Computing*, v. 1., p. 1-8, Rio de Janeiro, Brasil.

[13] Rizvi, S., Mendelzon, A., Sudarshan, S., Roy, P. (2004) Extending Query Rewriting Techniques for Fine Grained Access Control. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pp. 551-562, New York, USA.

[14] Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E. (1996) Role-based access control models. *IEEE Computer*, vol. 29, no. 2, pp 38-47..

[15] TPC Council, 2008. TPC Benchmark H Standard Specification Revision 2.8.0. Transaction Processing Perfermance Council. Disponível em <http://www.tpc.org/tpch/spec/tpch2.8.0.pdf>.

[16] Yang, L. 2009. Teaching database security and auditing. In Proceedings of the 40th ACM Technical Symposium on Computer Science Education, v.1, issue 1, pp. 241—245.