UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO

ESCOLA DE INFORMÁTICA APLICADA

CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Framework flexível para regras de autorização de acesso a dados

Nome do autor:

Sergio Gonçalves Puntar Filho

Nome do Orientador:

Leonardo Guerreiro Azevedo

Dezembro/2010

Framework flexível para regras de autorização de acesso a dados

Projeto de Graduação apresentado à Escola de Informática Aplicada da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) para obtenção do título de Bacharel em Sistemas de Informação

Nome do autor:

Sergio Gonçalves Puntar Filho

Nome do Orientador:

Leonardo Guerreiro Azevedo

Framework flexível para regras de autorização de acesso a dados

Aprovado em ____/_____/_____

BANCA EXAMINADORA

_____

Prof. Leonardo Azevedo, D.Sc. (UNIRIO)

_____

Prof. Fernanda Baião, D.Sc. (UNIRIO)

_____

Prof. Luiz Carlos Monte, D.Sc. (UNIRIO)

_____

Prof. Rodrigo Salvador Monteiro, D.Sc. (COPPE/UFRJ)

Rio de Janeiro, ____ de _____ de _____

_____

Sergio Gonçalves Puntar Filho

AGRADECIMENTOS

Agradeço primeiramente aos meus pais e minha irmã que sempre me apoiaram durante o meu desenvolvimento pessoal e profissional, e me possibilitaram chegar nesse ponto da minha vida.

Agradeço ao meu orientador, Leonardo Azevedo, pela sua dedicação, paciência e capacidade que foram essenciais para o desenvolvimento desse trabalho, e pela sua enorme contribuição tanto na minha formação acadêmica quanto na minha formação profissional.

Agradeço a todos que contribuíram de alguma forma para esse trabalho, em especial Clarissa Romeiro, Claudia Cappelli, Diego Duarte, Fernanda Baião, Léo Antunes, Raphael Melo e toda equipe da Petrobras envolvida.

Agradeço a todos os antigos e atuais integrantes do grupo de pesquisa NP2Tec que me acompanharam durante tantos projetos. As minhas "chefas" Gláucia Sá Fortes e Andréa Magalhães agradeço o ensinamento da importância do trabalho em equipe.

Agradeço o corpo docente da UNIRIO pelo conhecimento transmitindo sempre se esforçando para proporcionar o melhor aprendizado possível. Uma lembrança especial aos professores Flávia Santoro, Kate Revoredo e Márcio Barros por algumas das melhores aulas que já tive.

Agradeço aos meus colegas de graduação com os quais dividi momentos de felicidade e angústia durante quase cinco anos de estudo, em especial Alexandre Souza, Brunno Athayde, Diego Duarte, Felipe Klussmann, Paulo Castro e Thiago Yusuke.

Por fim agradeço a todos os meus amigos que fazem de mim uma pessoa mais feliz.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

API - *Application Programming Interface*

BRG - *Business Rules Group*

DAC - *Discricionary Access Control*

DBA - *Database Administrator*

DML - *Data Manipulation Language*

DoD - *Department of Defense*

ER - Entidade e Relacionamento

ERA - Execução de Regras de Autorização

FARBAC - *Flexible Approach for Role-Based Access Control*

GRA - Gestão de Regras de Autorização

IP - *Internet Protocol*

MAC - *Mandatory Access Control*

OLAP – *On-line Analytical Processing*

PL/SQL - *Procedural Language/Structured Query Language*

RBAC - *Role-based Access Control*

RDBAC - *Reflective Database Access Control*

SF - *Scale Factor*

SGA - *System Global Area*

SGBDs - Sistema de Gerência de Banco de Dados

SOA - *Service-oriented Architecture*

SOX - Sarbanes-Oxley

SQL - *Structured Query Language*

TD - *Transaction Datalog*

TPC - *Transaction Processing Performance Council*

UGA - *User Global Area*

VPD - *Virtual Private Database*

RESUMO

Esse trabalho propõe um *framework* flexível para gestão e execução de regras de autorização de acesso a dados sobre SGBDs relacionais, independentemente das aplicações que acessam a base de dados. O *framework* adota a abordagem RBAC para definição de políticas, e foi implementado no Sistema Gerenciador de Banco de dados Oracle. Dessa forma, a segurança de acesso aos dados é construída apenas no servidor de dados, em vez de em cada aplicação que acessa os dados, e é aplicada pelo próprio SGBD. Foram executados testes experimentais usando dados e consultas do Benchmark TPC-H. Os resultados indicam a eficácia da proposta.

Palavras-chave: Segurança em banco de dados, Segurança da informação, Regras de autorização de acesso a dados, VPD, Benchmark TPC-H.

Capítulo 1: INTRODUÇÃO

O objetivo deste capítulo é contextualizar o assunto abordado neste trabalho, bem como apresentar a motivação, seu objetivo e a estrutura dos capítulos.

1.1 Motivação

Uma regra de negócio é uma declaração que define ou restringe algum aspecto do negócio. Ela destina-se a definir a estrutura do negócio ou controlar ou influenciar o comportamento do negócio [BRG, 2000]. De acordo com as regras de taxonomia definidas pela BRG [2000], uma das categorias de regras de negócio é a regra assertiva de ação de autorização ou simplesmente regra de autorização.

Uma regra de autorização especifica uma prerrogativa ou privilégio para uma ou mais ações. A declaração da regra de autorização restringe a **quem** é permitido realizar uma **ação** na organização sobre quais **informações** [BRG, 2000].

Tanto as organizações privadas quanto as governamentais dependem fortemente de sistemas de informação para atender as suas necessidades táticas, estratégicas, operacionais, financeiras e requisitos de tecnologia da informação. A integridade, disponibilidade e confidencialidade de sistemas de software, bancos de dados, e redes de dados são grandes preocupações em todos os setores. A divulgação ou acesso não autorizado a recursos corporativos podem interromper as operações de uma organização, acarretando em sérios impactos financeiros, legais, de segurança humana, de privacidade pessoal e de confiança do público [Ferraiolo e Khun, 1992].

Regras de autorização são tradicionalmente implementadas nas aplicações através de mecanismos simples de controle de acesso (sessões de banco de dados, usuários e perfis) disponibilizados pela maioria dos SGBDs (Sistema de Gerência de Banco de Dados) [Jeloka *et al*., 2008]. Murthy e Sedlar [2007] salientam que esse é um mecanismo pesado e difícil de aplicar em outras camadas. Além disso, esses mecanismos não são suficientes para restringir o acesso a subconjuntos de dados. Por exemplo, considerando um sistema para vendas *online*, é necessário controlar que usuários vejam apenas suas próprias contas, ou ainda, é necessário restringir o acesso

de gerentes de departamentos locais somente aos pedidos realizados nas lojas que gerenciam e que possuem valor total inferior a R$1.000,00 (se, por exemplo, somente gerentes globais podem lidar com pedidos de alto valor).

Em geral, as aplicações tendem a definir suas próprias políticas de segurança e aplicá-las na camada de aplicação. Contudo, essa não é a solução ideal quando tratamos de cenários muito freqüentes onde as grandes empresas são obrigadas a aderir a um novo conjunto de regras de autorização definidas por iniciativas de controle derivados de SOX [SOX, 2009] ou outra demanda de segurança. Nesses casos, já existem várias aplicações acessando dados armazenados em bancos de dados legados, o que torna o controle de permissões um problema muito complexo de se lidar.

Existe, portanto, a necessidade de um *framework* flexível para implementação de regras de autorização em bancos de dados já existentes que minimize as mudanças necessárias em aplicações legadas que manipulam estes dados.

1.2 Objetivo

O objetivo desse trabalho é propor um *framework* flexível para gerenciamento e execução de regras de autorização em SGBDs relacionais que afete minimamente as aplicações que acessam o banco de dados.

Este *framework* é composto por dois componentes: gestão de regras de autorização e execução de regras de autorização. Neste trabalho, além da proposta do *framework*, é definido e implementado um modelo flexível para execução de regras de autorização.

O componente de execução de regras de autorização adota a abordagem de definição de políticas RBAC (*Role-based access control*), e foi implementado no SGBD Oracle 10g utilizando a funcionalidade VPD (*Virtual Private Database*).

Nesta proposta, a segurança de acesso aos dados é construída apenas uma vez no servidor de dados, em vez de em cada aplicação que acessa os dados, e é aplicada pelo próprio servidor de banco de dados, não importando o método utilizado para acessar os dados. Os resultados experimentais comprovaram a eficiência da

abordagem proposta em um cenário construído sobre a especificação do Benchmark TPC-H para bancos de dados [TPC *Council*, 2008].

1.3 Estrutura da monografia

Este trabalho está dividido da seguinte forma. O presente capítulo apresenta a motivação, objetivo e contextualiza o assunto deste trabalho. O capítulo 2 apresenta os conceitos de segurança em bancos de dados e trabalhos relacionados. O capítulo 3 apresenta a proposta de *framework* para autorização e gestão de regras de autorização, enquanto que o capítulo 4 apresenta a implementação da proposta. O capítulo 5 apresenta a avaliação experimental da proposta. Por fim, o capítulo 6 apresenta a conclusão.

Capítulo 2: SEGURANÇA EM BANCO DE DADOS

O controle de acesso a dados é um dos principais componentes da segurança de banco de dados, e tem sido o foco de muitas iniciativas de segurança de dados. Funcionalidades de controle de acesso vêm sendo fornecidas como importantes ferramentas de gerenciamento de banco de dados por muitos fornecedores de SGBDs. Geralmente, esses mecanismos de segurança consistem na definição de usuários (que são identidades no nível de banco de dados para todos que se conectam ao banco), perfis (que agrupam usuários com privilégios similares às estruturas e aos dados do banco) e sessões de banco de dados (que permitem aos SGBD identificar e controlar a seqüência de comandos executados por um usuário em cada conexão com a base de dados) [Murthy e Sedlar, 2007].

Murthy e Sedlar [2007] argumentam que esses mecanismos de controle de acesso, apesar de simples e freqüentemente utilizados, apresentam inconvenientes significantes. Em primeiro lugar, o mesmo dado sensível pode precisar ser acessado por várias aplicações diferentes, e é difícil conciliar os mecanismos de segurança das diferentes aplicações. Em segundo lugar, muitas ferramentas de mineração de dados, *data warehousing*, etc., requerem acesso direto ao banco de dados (via SQL), e é impossível aplicar segurança no nível de aplicação durante acessos diretos em SQL. Em terceiro lugar, a falta de um *framework* de segurança comum torna difícil o gerenciamento das políticas, o que aumenta o risco de falhas de segurança. Por fim, há um impacto significante no desempenho ao sempre avaliar as regras de autorização de acesso na camada de aplicação.

Os mecanismos de controle de autorização de acesso podem ser classificados em DAC (Discricionary Access Control), MAC (Mandatory Access Control), ambos propostos por [DoD 1983], e RBAC (Role-Based Access Control) [Ferraiolo e Khun, 1992].

Políticas DAC (*Discricionary Access Control*) restringem acesso a objetos baseados na identidade dos usuários (e/ou grupos aos quais eles pertencem) e em regras de acesso que definem o que os usuários podem ou não acessar. As políticas

DAC podem ser implementadas em um modelo de matriz de acesso, que indica os privilégios que um usuário tem sobre cada objeto. Nesse caso o objeto pode ser uma tabela, visão, *procedure* ou qualquer outro objeto do banco de dados. O modelo de matriz de acesso pode ser implementado através de uma tabela de autorização, de uma lista de controle de acesso ou de uma lista de possibilidades (*capability list*) geralmente utilizada por SGBDs.

Yang [2009] aponta que políticas DAC não garantem controle sobre o fluxo de informações, tornando possível que informações cheguem a usuários que não têm permissão de lê-las. O Microsoft SQL Server, o MySQL, o Oracle, o DB2 e o Sybase implementam políticas DAC através de modelos de matriz de acesso [Yang, 2009].

Políticas MAC (*Mandatory Access Control*), também conhecidas como *label security*, são baseadas em regulamentações mandatórias determinadas por uma autoridade central [Yang, 2009]. A forma mais comum de MAC é a política de segurança de múltiplos níveis usando classificação de sujeitos (usuários ou grupos) e objetos (dados) dos sistemas, onde uma classe de acesso (rótulo) é concedida para cada objeto e sujeito. Uma classe de acesso consiste de dois componentes: um nível de segurança e um conjunto de categorias. O nível de segurança é um elemento de um conjunto ordenado hierarquicamente, como, por exemplo: Altamente Secreto (AS), Secreto (S), Restrito (R) e Público (P), onde AS > S > R > P. O conjunto de categorias é um subconjunto de um conjunto não ordenado, cujos elementos refletem áreas funcionais.

Nas políticas MAC, os rótulos são atribuídos às linhas inteiras das tabelas, dessa forma não é possível definir regras de autorização para um subconjunto de atributos da linha. A atual implementação de políticas MAC no Oracle [Levinger *et al.*, 2003] requer que uma nova coluna seja criada para cada rótulo, assim, se existirem muitas regras de autorização e perfis de usuário referenciando a mesma tabela, o espaço extra ocupado pode se tornar um problema. Além disso, essa implementação é pouco flexível, pois mudanças em uma regra requer que todas as tabelas onde a regra está implementada sejam atualizadas.

As políticas MAC controlam o fluxo de informações, embora não abordem as ações que podem ser executadas pelos sujeitos sobre os dados [Ferraiolo e Khun,

1992]. A maioria dos fornecedores de SGBDs oferece ferramentas de suporte a MAC, como o Oracle, o Sybase e o Microsoft SQL Server [Yang, 2009].

Políticas RBAC (*Role-Based Access Control*) fundamentam as decisões de controle de acesso sobre as funções que um usuário tem permissão de realizar dentro da organização [Ferraiolo e Khun, 1992]. Uma política RBAC estabelece que ações e assuntos são permitidos para um usuário que possui determinado papel. Organizações tipicamente utilizam a RBAC para especificar e aplicar políticas de segurança específicas, de forma que sejam naturalmente mapeadas na estrutura organizacional [Yang, 2009]. Nesse contexto, políticas baseadas em papéis regulam o acesso aos dados baseadas nas responsabilidades organizacionais de cada usuário, pois o interesse principal é proteger a integridade da informação: **quem** pode realizar qual **ação** sobre qual **informação** [Ferraiolo e Khun, 1992].

Políticas baseadas em papéis quebram a autorização de um usuário em diversas atribuições de papéis, e diversas atribuições de permissões aos papéis. Tanto o Microsoft SQL Server quanto o Oracle suportam o conceito de papéis [Yang, 2009]. O Oracle implementa o RBAC através da ferramenta VPD (*Virtual Private Database*) [Jeloka *et al*., 2008]. O VPD acrescenta predicados (cláusulas *WHERE*) dinamicamente a qualquer instrução SQL executada sobre uma tabela protegida, assim modificando dinamicamente o acesso aos dados pelo usuário. Em outras palavras, as regras são armazenadas como predicados e, quando o usuário executa uma operação sobre uma tabela, o predicado é utilizado para restringir o acesso.

Políticas RBAC são mais flexíveis que as políticas MAC e, portanto, são o foco do *framework* proposto. No entanto, a definição e aplicação de políticas RBAC não é um trabalho simples em cenários reais, pois exige muito esforço e conhecimento por parte do responsável por criar e atribuir regras e papéis, geralmente um DBA (*Database Administrator*). No Oracle, por exemplo, para cada definição de regra em uma política RBAC, uma nova função PL/SQL deve ser criada para implementar a política.

Existe, portanto, a necessidade de um mecanismo flexível e fácil de usar para ajudar os administradores a definir e gerenciar políticas RBAC. Esse mecanismo

deveria também ser configurável dinamicamente e fácil de implementar, por exemplo no caso do Oracle, usando a funcionalidade VPD já existente no SGBD.

Outros trabalhos relacionados a essa área já foram realizados, como o *framework* para definição de políticas de controle de acesso refletivas no banco de dados (*Reflective Database Access Control* - RDBAC) apresentado por [Olson *et al.*, 2008], que propõe a definição de regras reflexivas (que se baseiam em informações já presentes no banco de dados) através do uso de *Transaction Datalog* (TD). Essa proposta foi implementada no Microsoft SQL Server por [Olson *et al.*, 2009] que comprovou a sua eficiência. O *framework* apresentado nesse trabalho utiliza o mesmo conceito de regras reflexivas, apesar de armazenar essas regras diretamente em SQL e não em TD.

Outro trabalho interessante é o apresentado em [Giuri e Iglio, 2009] que propõe a definição de modelos de papéis para RBAC. Estes modelos provêem mecanismos para definição de políticas de controle de acesso baseadas em conteúdo, através do conceito de privilégios parametrizados para restringir subconjuntos de dados. A definição de perfis do *framework* proposto funciona de forma parecida, onde em uma hierarquia de perfis os perfis superiores definem as regras e os parâmetros e os perfis inferiores definem os valores para os parâmetros.

Capítulo 3: PROPOSTA DE *FRAMEWORK* FLEXÍVEL PARA GESTÃO E EXECUÇÃO DE REGRAS DE AUTORIZAÇÃO

Esse capítulo apresenta a proposta de framework flexível para gestão e execução de regras de autorização em bancos de dados corporativos, também denominado de FARBAC (*Flexible Approach for Role-Based Access Control*) [Azevedo *et al*., 2010]. O framework é composto de dois módulos: (i) Gestão de Regras de Autorização (GRA) e (ii) Execução de Regras de Autorização (ERA).

O módulo de gestão de regras de autorização é responsável pela criação, mudança, visualização, composição, teste e simulação das regras. Todas as regras de autorização definidas nesse módulo são armazenadas em um banco de dados de regras de negócio. As regras, que são definidas como cláusulas WHERE, serão dinamicamente adicionadas pelo ERA aos comandos executados pelas aplicações. Dessa forma, as regras se baseiam em informações já presentes no banco de dados e que serão avaliadas pela cláusula WHERE definida. Esse tipo de regra também é denominado regra reflexiva, de modo que o FARBAC pode ser classificado também como um RDBAC [Olson *et al.*, 2008].

O módulo de execução de regras de autorização segue o modelo de políticas RBAC descrito no capítulo 2, e é responsável por garantir a confidencialidade dos dados em tempo de execução. O ERA garante que as regras de autorização previamente definidas sejam aplicadas durante a execução de cada aplicação que acessa o banco de dados corporativo. Em outras palavras, o módulo de execução controla todas as aplicações, de forma que os dados recuperados por estas aderem às regras armazenadas no banco de dados de regras de negócio. Para evitar mudanças no código-fonte de aplicações legadas, o módulo ERA deve ser implementado em um componente independente das aplicações acessando o SGBD, por exemplo, como um *middleware*.

Definir e executar regras de autorização são tarefas realizadas em dois momentos distintos e, idealmente, por duas equipes distintas. Tipicamente, as funções do GRA seriam utilizadas pelos usuários de negócio, enquanto as funções do

ERA seriam monitoradas pelos usuário de TI. Primeiramente, os usuários de negócio da equipe de definição de regras de autorização utilizariam o módulo GRA para definir e criar o conjunto de regras que refletem as políticas RBAC sobre os dados corporativos. Através do módulo GRA, deve ser possível acessar o esquema conceitual da organização, que representa uma definição do negócio em alto nível de todos os conceitos e relacionamentos armazenados no banco de dados, para cadastrar regras de autorização.

Para definir uma nova regra de autorização para uma política RBAC, o usuário do módulo GRA deve definir um perfil (por exemplo, *GerenteLocal*), escolher o conceito a ser controlado (por exemplo, *Pedidos*), cada atributo do conceito que será usado para restringir o acesso (por exemplo, *ValorTotal*), juntamente com o intervalo de valores aceitos (por exemplo, *ValorTotal < 1.000,00*), e a operação de deve ser controlada (*Select*, *Insert*, *Update*, *Delete*). O módulo GRA deve, portanto, fornecer uma interface gráfica que permita que os usuários de negócio definam e gerenciem as suas próprias regras de autorização e políticas RBAC, como exemplifica a regra na Figura 1. O lado esquerdo da Figura 1, são apresentados conceitos que podem ser considerados nas regras. O lado direito apresenta os perfis existentes. A parte inferior da figura ilustra as operações que podem ser aplicadas sobre os conceitos. No centro, estes aspectos são ligados a fim de definir a regra que especifica uma condição sobre um conceito (*Pedidos.ValorTotal < 1.000,00*) que deve ser aplicada quando um usuário com determinado perfil (*GerenteLocal*) executa uma operação (*Select* ou *Delete*).

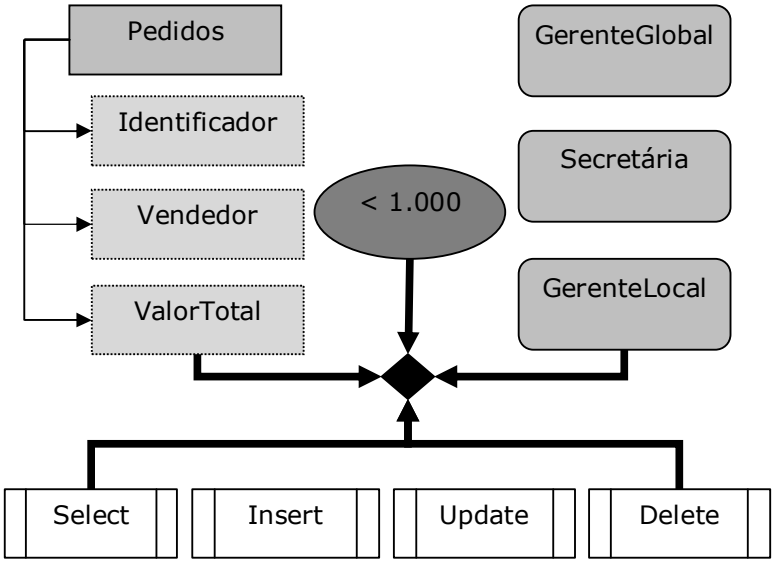

Figura 1 - Estrutura de uma Regra de Autorização

Em tempo de execução (Figura 2), o módulo ERA captura cada conexão das aplicações clientes com o banco de dados corporativo e transforma o comando SQL (digamos, *SQL*) em um comando modificado (digamos, *SQL'*), de acordo com as regras de autorização presentes no banco de dados de regras de negócio na forma de cláusulas WHERE e de acordo com o usuário. O módulo ERA então envia o comando *SQL'* para o SGBD e remete os dados retornados para a aplicação cliente. Observe que uma premissa para o funcionamento do ERA é que ele seja capaz de identificar o usuário que está realizando a consulta. Em outras palavras, para que o ERA aplique o controle de acesso com sucesso, deve-se garantir que a identidade do usuário seja propagada corretamente nas diversas camadas do sistema até o módulo ERA.

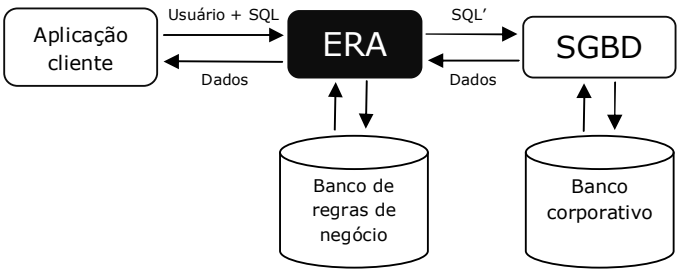

Figura 2 - Arquitetura do módulo de Execução de Regras de Autorização

Essa abordagem de modificação de consulta, também conhecida como *Truman Model* [Rizvi *et al*., 2004], provê a cada usuário (ou grupo de usuários) uma visão pessoal e restrita da base de dados. As consultas são modificadas de forma transparente para que o usuário não veja nada mais do que a sua própria visão da base. Isso pode causar inconsistências entre o que o usuário espera ver e o que o sistema retorna. Para evitar mal entendidos os usuários devem conhecer muito bem a sua visão da base de dados.

Em outra abordagem, conhecida como *Non-Truman Model*, a consulta original é submetida a um teste de validação, onde ela é dita válida somente se puder ser respondida utilizando apenas as informações disponíveis para o usuário. Ao falhar a validação, a consulta é rejeitada e o usuário é notificado sobre isso. Várias desvantagens podem ser apontadas nesse modelo, incluindo obrigar os usuários a formular consultas válidas, e fornecer respostas insuficientemente descritivas quando uma consulta não é permitida [Olson *et al*., 2008].

A seguir, a seção 3.1 apresenta o metamodelo para armazenamento das regras de autorização e a seção 3.2 apresenta o algoritmo para construção do predicado de autorização.

3.1 Modelo (ER) para regras de autorização

As regras de autorização são armazenadas no banco de dados de regras de negócio de acordo com o modelo conceitual de entidades e relacionamentos apresentado na Figura 3 e descrito a seguir.

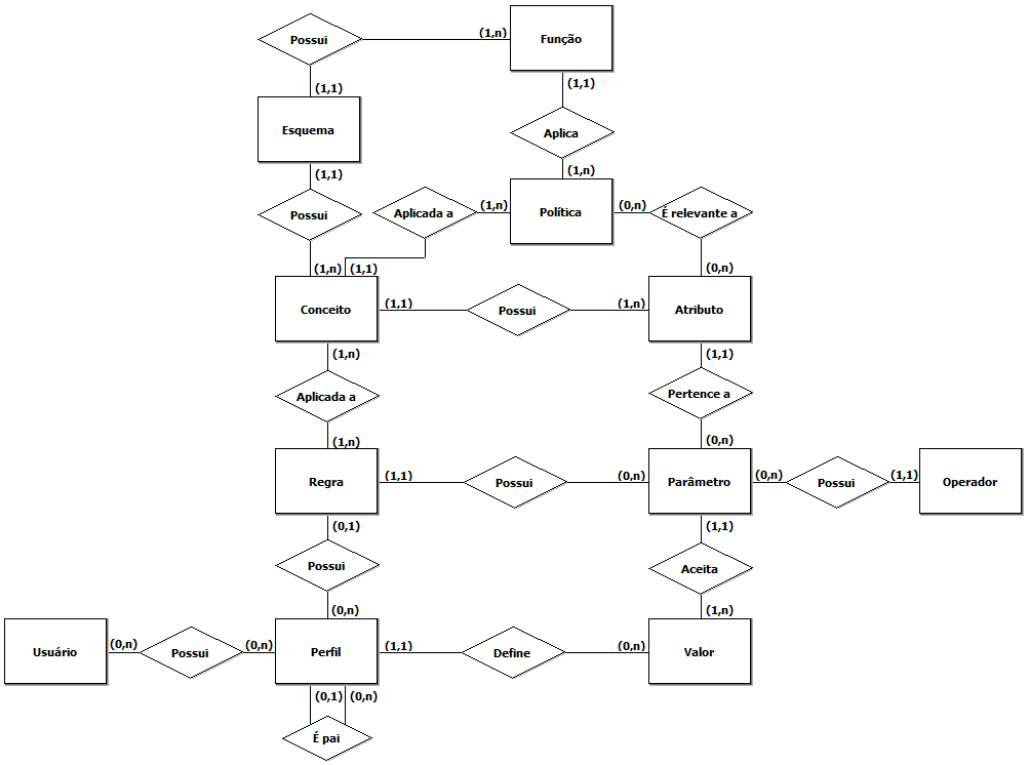

Figura 3 - Modelo de entidade e relacionamento do *framework*

Os *Usuários* são agrupados em *Perfis*, onde cada usuário pode ser associado a diversos perfis, e cada perfil pode agrupar diversos usuários. O auto-relacionamento da entidade *Perfil* permite a representação de uma hierarquia de perfis. Por exemplo, os perfis de *Gerente de Vendas da América e Ásia* e de *Gerente de Vendas da Europa* podem ser especializações do perfil *Gerente de Vendas* (Figura 4). Isso significa que regras de autorização atribuídas a um perfil mais geral (*Gerente de Vendas*) devem também ser aplicadas a todos os seus descendentes. Além disso, perfis descendentes podem determinar valores específicos para os parâmetros do predicado. Por exemplo, o perfil *Gerente de Vendas* pode restringir o acesso aos pedidos baseado no atributo *continenteDeOrigem*; o perfil *Gerente de Vendas da América e Ásia* pode então especificar o(s) valor(s) para o atributo controlado (por exemplo, "*America, Asia"*). Dessa forma é possível que a política RBAC implemente a regra, restringindo o acesso do perfil *Gerente de Vendas América e Ásia* somente para os pedidos onde *continenteDeOrigem in (America, Asia)*.

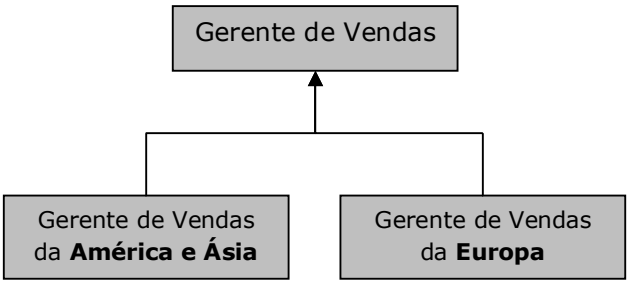

Figura 4 - Hierarquia de Gerentes de Vendas

A entidade *Regra* representa as definições de regras que são armazenadas como cláusulas SQL, que serão usadas para transformar os comandos SQL provenientes da aplicação cliente. Uma cláusula restringe o acesso do *Usuário* a um subconjunto dos dados de um *Conceito,* que pode ser uma tabela, uma visão ou um sinônimo. Portanto, existe um relacionamento *Regra* × *Conceito*. A mesma cláusula SQL pode ser usada para definir regras de autorização em diferentes conceitos, e cada *Conceito* pode possuir mais de uma *Regra*.

Um *Parâmetro* denota uma expressão no formato "Atributo Operador Valor" relacionando as entidades correspondentes (*Atributo* × *Operador* × *Valor*). A entidade *Atributo* representa um atributo de um conceito do banco de dados utilizado para restrição de acesso. A entidade *Operador* representa um operador binário (incluindo =, <>, >, >=, <, <=, IN e NOT IN). A entidade *Valor* representa um valor (ou lista de valores) dentro do domínio do atributo, que delimitam um subconjunto de dados. O relacionamento *Conceito* × *Atributo* indica de que conceito um atributo pertence da mesma forma que o relacionamento *Esquema* × *Conceito* indica de que esquema o conceito pertence. Esse relacionamento denota uma representação em alto nível do esquema do banco de dados para o qual as regras de autorização são aplicadas.

A *Política* dita o controle de acesso que deve ser aplicado a um conceito. Uma política de controle de acesso é aplicada por uma *Função* (que também pertence a um *Esquema*). Durante sua execução, esta função identifica o *Conceito* sendo acessado e o *Usuário* executando o comando. Ela captura os dados das outras tabelas do modelo para construir o predicado da regra de autorização (ou predicado de

autorização) daquele conceito para aquele usuário. Além disso, o relacionamento *Política* × *Atributo* determina quais atributos do conceito possuem dados sensíveis.

O mesmo usuário pode possuir mais de um perfil relacionado a um mesmo conceito. Dessa forma, em tempo de execução, mais de uma regra (predicado de autorização) será retornada e, portanto, devem ser compostas de alguma forma. Essa composição pode ser feita através do uso dos operadores OR ou AND. A composição por OR, que chamamos de composição permissiva, permite que o usuário tenha acesso à união dos subconjuntos de dados permitidos por cada regra. A composição por AND, que chamamos de composição restritiva, limita o acesso do usuário à intercessão dos subconjuntos de dados permitidos por cada regra. Por exemplo, um usuário que possua os perfis *Gerente de Vendas da Europa* e *Gerente de Vendas da França*, com a composição permissiva teria acesso as dados de toda a Europa, já com a composição restritiva o seu acesso seria apenas aos dados da França, como mostra a Figura 5.

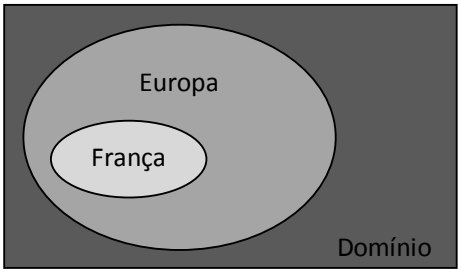

Figura 5 - Exemplo de composição de perfis

Tanto a composição permissiva quanto a composição restritiva são suportadas pelo *framework* proposto, e é dever da organização decidir que tipo usar. Nos testes experimentais desse trabalho foi escolhida a composição permissiva.

3.2 Algoritmo para construção do predicado de autorização

O *framework* proposto possui uma função genérica responsável por construir, em tempo de execução, o predicado de autorização para cada conceito. Essa função é aplicada a todas os conceitos com dados sensíveis e, de acordo com o usuário

realizando o acesso e com os dados cadastrados no modelo de armazenamento de regras de autorização (Figura 3), retorna o predicado de autorização.

Ao aplicar uma regra de autorização, o *framework* recebe a identificação do conceito sendo acessado e executa os passos apresentados na Figura 6.

```
Recupera o usuário que está executando a consulta
A partir do relacionamento Usuário × Perfil, recupera os
perfis do usuário
Para cada perfil:
   A partir do relacionamento Perfil × Regra × Conceito,
   identifica as regras que devem ser aplicadas de acordo com
   o perfil pai da hierarquia
   Para cada regra:
      A partir do relacionamento Regra × Parâmetro, identifica
      os parâmetros específicos da regra
      Para cada parâmetro:
         A partir do relacionamento Perfil × Valor, recuperar
         o(s) valor(s) aceito(s) pelo parâmetro para filtragem
         dos dados
Finalmente, o predicado de autorização é montado e retornado
```

Figura 6 – Passos para construção dos predicados das regras

Capítulo 4: IMPLEMENTAÇÃO DO *FRAMEWORK* PARA EXECUÇÃO DE REGRAS DE AUTORIZAÇÃO

Nesse trabalho o módulo ERA da proposta foi implementado através do *Virtual Private Database* (VPD) do SGBD Oracle versão 10g. O VPD é uma abordagem RBAC implementada pela Oracle, que foi estendida para acomodar o *framework* proposto. Esse capítulo apresenta uma descrição do VPD, bem como a implementação do modelo de armazenamento de regras de autorização e da função genérica para retornar o predicado de autorização.

4.1 Virtual Private Database

O Virtual Private Database (VPD) permite reforçar a segurança diretamente em tabelas, visões e sinônimos, anexando a estes elementos as políticas de segurança, que são automaticamente ativadas sempre que um usuário tenta acessar dados. Como a política de segurança é aplicada diretamente à base de dados, essa tecnologia obedece ao pré-requisito do *framework* de estratificar as aplicações clientes da execução da regra.

Quando o usuário tenta acessar (direta ou indiretamente) uma tabela, visão, ou sinônimo protegido por uma política do VPD (implementada em uma função PL/SQL), o servidor altera dinamicamente o comando SQL do usuário. Essa alteração cria uma condição para cláusula *WHERE*, também conhecida como predicado VPD, que é retornada pela função que implementa a política de segurança. As funções de políticas do VPD podem fazer chamadas a outras funções e, através de pacotes PL/SQL, é possível incluir também chamadas em C ou em Java.

As políticas do VPD podem ser definidas para os comandos *Select*, *Insert*, *Update*, *Delete* e *Index*. No caso dos comandos *Insert* e *Update*, é possível que a política seja aplicada também ao resultado dos comandos, impedindo que um usuário insira ou atualize uma informação para um estado que ele não possua acesso. Por exemplo, um usuário com o perfil *Gerente de Vendas da América e Ásia* não poderia

incluir um pedido proveniente da França, nem alterar a origem de um pedido já existente para França.

As políticas VPD podem ainda ser aplicadas considerando colunas. Este caso só pode ser aplicado em tabelas e visões. Ao especificar as colunas sensíveis de uma tabela, a política é aplicada somente quando essas colunas são referenciadas. Por exemplo, a tabela Empregados possui uma política protegendo a coluna *matricula*, onde cada usuário pode visualizar apenas a sua própria matrícula. A função da política retorna o predicado *matricula = 'minha_matricula'*. O usuário faz então a seguinte consulta na tabela:

```
SELECT nome FROM Empregados;
```

Como a coluna *matricula* não foi referenciada na consulta, a política não é aplicada e a consulta retorna todos os registros, pois se entende que não há nenhum dado sensível a ser protegido. Contudo, se a consulta referenciar a coluna *matricula*:

```
SELECT nome, matricula FROM Empregrados;
```

Então o VPD reescreve o comando da seguinte forma:

```
SELECT nome, matricula FROM Empregrados
WHERE matricula='minha_matricula';
```

Nesse caso somente o registro do próprio usuário será retornado. Adicionalmente, é possível mascarar a coluna sensível de forma que, mesmo referenciado essa coluna, a consulta retornaria todos os registros, porém as matrículas que o usuário não tem acesso seriam substituídas pelo valor *null*.

Logo, as políticas criadas no VPD podem se enquadrar nos seguintes casos:

- **Regra de autorização sobre o objeto:** Neste caso é necessário definir sobre qual tabela, visão ou sinônimo a restrição será aplicada. Toda vez que o objeto aparecer na cláusula *FROM* de uma operação SQL ou DML (por padrão ela é aplicada para todas os comandos, *Select*, *Insert*, *Update*, *Delete* e *Index*), que tenha o acesso controlado, a regra será aplicada.

- **Regra de autorização sobre colunas:** Neste caso é necessário definir sobre qual tabela (ou visão) e sobre quais colunas da tabela (ou visão) a regra de autorização deve ser aplicada. Por padrão, a regra de autorização é aplicada a todas as colunas. Ou seja, caso não seja informada nenhuma coluna, a regra é aplicada a qualquer operação executada sobre a tabela (ou visão).

  Portanto, no caso da regra de autorização sobre colunas, a autorização será aplicada somente quando as colunas sensíveis estiverem envolvidas na operação. Por outro lado, se estas colunas não forem envolvidas na operação, mesmo estando em uma tabela protegida, o usuário terá acesso aos dados da tabela.

- **Regra de autorização sobre colunas com mascaramento:** Este caso é semelhante ao caso anterior (Regra de autorização sobre colunas). A única diferença está no fato de que quando a regra de autorização é aplicada, são retornados todos os registros que atendem a consulta original (sem a inclusão do predicado VPD), porém as informações que o usuário não tem acesso são mascaradas com o valor *null*.

4.1.1 Oracle Application Context

O *Oracle Application Context* é uma funcionalidade que ajuda o VPD a aplicar as políticas de segurança de acordo com o usuário conectado. O Oracle provê um *namespace* de contexto padrão, chamado *userenv*, que permite acessar atributos pré-definidos. Esses atributos são informações de sessão que o SGBD automaticamente captura para cada sessão. Por exemplo, o endereço IP da origem da conexão, o nome do usuário conectado, o nome do usuário de *proxy* (nos casos em que a conexão passa por uma camada intermediária), entre outros.

Cada aplicação pode possuir seu próprio *namespace* de contexto específico, cujos atributos podem ser acessados pelas funções que implementam as políticas VPD. Uma função de política do VPD pode retornar diferentes predicados para cada usuário, grupo de usuários ou aplicação, de acordo com os valores dos atributos dos contextos. O contexto de aplicação permite acesso seguro aos atributos utilizados como base para as políticas. Por exemplo, o usuário com o atributo de posição

"Gerente" possui uma política de segurança diferente de um usuário com o atributo de posição "Empregado".

4.1.2 Tipos de políticas VPD

Quando uma política VPD é criada, é possível estabelecer eficiências de tempo de execução ao definir se a política é estática, compartilhada, sensível ao contexto ou dinâmica. A Tabela 1 lista os tipos de políticas e a relação com o predicado e com a função de cada uma delas.

Tabela 1 – Tipos de política VPD e suas relações com os predicados e funções

Tipo de política	Predicado e função da política
Estática	Mesmo predicado para qualquer usuário acessando o objeto
Estática compartilhada	Mesmo que a estática, mas a política pode ser compartilhada por múltiplos objetos
Sensível a contexto	A função de construção do predicado é re-executada sempre que uma mudança no contexto é detectada
Sensível a contexto compartilhada	Mesmo que a sensível a contexto, mas a política pode ser compartilhada por múltiplos objetos
Dinâmica	A função de construção do predicado sempre é executada cada vez que a política é aplicada

A função da política estática é executada apenas uma vez e seu resultado é armazenado na *System Global Area* (SGA), que é uma área de memória compartilhada por todo o banco de dados. Qualquer acesso realizado por um usuário nesse objeto utilizará esse mesmo predicado. Da mesma forma, a função da política estática compartilhada é executada apenas uma vez e seu resultado é armazenado na SGA. Contudo, antes de executar a função, o VPD verifica se o predicado dessa política já foi gerado por outro objeto, nesse caso aproveitando esse predicado.

A função da política sensível a contexto é executada uma vez e seu resultado é armazenado na memória de sessão, chamada de *User Global Area* (UGA), que é única para cada sessão do banco de dados. Esse mesmo predicado será reutilizado durante essa sessão, até que uma mudança no contexto seja detectada. Nesse caso, a função é re-executada e o novo predicado é armazenado na memória de sessão. Analogamente à política estática compartilhada, a política sensível a contexto compartilhada verifica se o predicado de uma política já foi gerado por outro objeto antes de executar a função.

Por fim, a função da política dinâmica é executada toda vez que a política é aplicada a um objeto. Esse é o tipo padrão de política do VPD, e foi o tipo utilizado para os testes experimentais desse trabalho.

4.2 Modelo (relacional) para regras de autorização

A seguir é apresentado o modelo relacional para regras de autorização utilizado para a implementação da proposta de *framework* flexível (Figura 7). O modelo é genérico o suficiente para ser utilizado para controle de autorização em diferentes bases de dados.

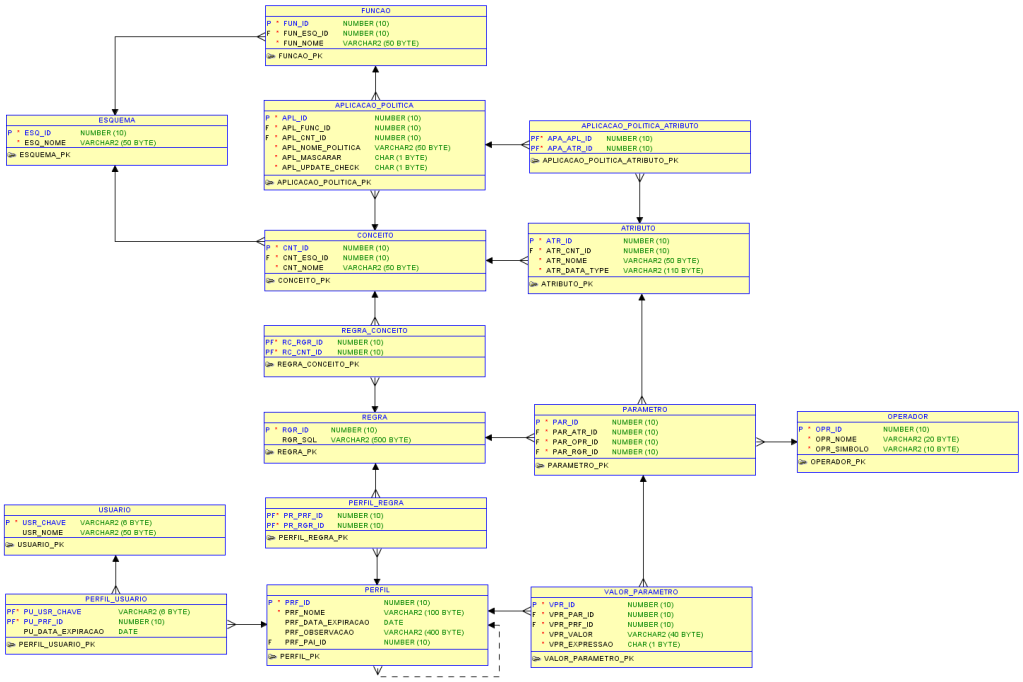

Figura 7 - Modelo relacional da implementação do *framework*

As seguintes tabelas fazem parte do modelo relacional para armazenar as regras de autorização:

- **USUARIO**: Essa tabela armazena os usuários que serão associados aos perfis de acesso à base de dados. Nessa implementação, os usuários cadastrados nessa tabela devem ser usuários do banco de dados. Isto porque, durante a execução da regra de autorização, a função recupera o usuário (usuário do banco de dados) que está executando a operação e, a partir deste usuário, obtém os perfis aos quais ele está associado. Existem outras formas de se recuperar o usuário realizando o acesso, contudo estas outras abordagens estão fora do escopo deste trabalho. A operação só é executada se um dos perfis do usuário tiver acesso para executá-la.

- **PERFIL**: Essa tabela armazena os perfis utilizados para aplicar as regras de autorização. A implementação atual contempla, através de um auto-relacionamento, a relação hierárquica de perfis em um único nível. No entanto, é possível estendê-la para contemplar mais níveis. Os níveis internos são utilizados para definir regras que se aplicam aos filhos. Dessa forma, na implementação atual, os perfis pais devem manter o campo do auto-relacionamento vazio,

enquanto os perfis filhos devem preenchê-lo com o identificador do perfil pai. É possível ainda criar um perfil simples, ou seja, que não é pai nem filho de nenhum outro perfil. Para isso, basta preencher o campo de auto-relacionamento com o identificador do próprio perfil.

- **PERFIL_USUARIO**: Essa tabela forma o relacionamento M × N entre as tabelas *USUARIO* e *PERFIL*, de forma que um usuário pode possuir vários perfis e cada perfil pode ser associado a vários usuários.
    - ♦ Essa tabela além de realizar o relacionamento M × N, também define a validade da aplicação de um perfil a um determinado usuário, através do atributo *PU_DATA_EXPIRACAO*. O perfil do usuário é válido enquanto esta coluna estiver com valor nulo ou, caso esteja preenchida, a data atual for menor que a data que ela armazena.
- **REGRA**: Essa tabela armazena as definições de regras que deverão ser aplicadas a cada perfil. Nesse modelo as regras são armazenadas como cláusulas SQL que serão parte do predicado retornado pela função da política do VPD. A regra não depende apenas do perfil do usuário que está executando a consulta, mas também do conceito sobro o qual essa consulta está sendo realizada. Como cada regra deve ser associada à dupla perfil e conceito, existe a possibilidade de um mesmo perfil ter diferentes regras para diferentes conceito. Analogamente, existe a possibilidade de um conceito estar associado a diversas regras de diversos perfis diferentes. A solução para esse problema é resolvido pela utilização das três tabelas descritas a seguir.
    - ♦ Em uma hierarquia de perfis, a regra associada ao perfil pai será cadastrada nessa tabela, e todos os perfis filhos vão depender dela. Dessa forma, os perfis filhos não possuirão entradas nessa tabela, mas sim nas tabelas de valores de parâmetros específicos, que serão discutidas nos itens abaixo.
- **CONCEITO**: Essa tabela armazena os nomes dos conceitos que possuem políticas de acesso, e ainda os nomes dos conceitos que possuem algum atributo relacionado a algum parâmetro de alguma regra específica. Ela possui um relacionamento com a tabela *ESQUEMA* que indica o esquema do qual o conceito pertence.

- **PERFIL_REGRA**: Essa tabela forma o relacionamento M × N entre as tabelas *PERFIL* e *REGRA* de forma que um perfil pode possuir diversas regras de autorização, e uma mesma regra possa ser associada a diversos perfis.
- **REGRA_CONCEITO**: Essa tabela forma o relacionamento M × N entre as tabelas *CONCEITO* e *REGRA* de forma que um conceito pode possuir diversas regras de autorização, e uma mesma regra possa ser associada a diversos conceitos.
    - ♦ O relacionamento M × N entre as tabelas *CONCEITO* e *REGRA* é interessante no caso em que existam vários conceitos com informações sobre as quais se deseja aplicar uma regra de autorização de um mesmo perfil. Por exemplo, os conceitos NACAO, EMPREGADO e CLIENTE podem ter uma mesma coluna *NACAO_ID* que armazena a id da nação, no primeiro conceito sendo uma chave primária e nas outros sendo uma chave estrangeira. Caso se deseje cadastrar uma regra que restrinja o acesso a determinadas nações para um determinado perfil (por exemplo, *NACAO_ID = 1*), esta regra poderia ser cadastrada para todos os três conceitos.
    - ♦ Observe que agora as tabelas *PERFIL*, *PERFIL_REGRA*, *REGRA*, *REGRA_CONCEITO* e *CONCEITO* permitem que um mesmo perfil tenha diferentes regras para diferentes conceitos, e ainda formam a dupla perfil e conceito discutida anteriormente.
- **PARAMETRO**: Essa tabela armazena os parâmetros das regras cadastradas na tabela *REGRA*, que serão os parâmetros específicos dos perfis filhos. Os parâmetros são associados à regra cadastrada na tabela *REGRA* para o perfil pai, indicando a qual regra o parâmetro se refere. No entanto, os valores para os parâmetros são associados aos perfis filhos, indicando qual(s) valor(s) deve(m) ser usado(s) para cada perfil filho. Por exemplo, o parâmetro que define a região está associado ao perfil *Gerente de Vendas*, mas os valores para os parâmetros estão associados aos perfis *Gerente de Vendas da América e Ásia* e *Gerente de Vendas da Europa*. Um parâmetro é formado por um atributo, um operador e o(s) valor(s) do parâmetro.

- Exemplo: *nome_regiao* in (‘América’, ‘Ásia’)
  Formato: *atributo* operador valor
- Observe que para todos os perfis filhos de uma hierarquia o atributo e o operador do parâmetro serão os mesmos, a variação é apenas do valor definido para o parâmetro por cada perfil filho.

- **ATRIBUTO**: Essa tabela armazena os atributos relacionados a um parâmetro de uma regra de acesso. A tabela atributo possui uma relação com a tabela *CONCEITO*, pois um atributo é um campo presente em um conceito.
- **OPERADOR**: Essa tabela que armazena os operadores binários existentes na SQL e que podem se utilizados para a definição de um parâmetro, são eles:
  - ‘=’ (igual)
  - ‘<>’ (diferente)
  - ‘>’ (maior)
  - ‘>=’ (maior ou igual)
  - ‘<’ (menor)
  - ‘<=’ (menor ou igual)
  - ‘IN’
  - ‘NOT IN’
- **VALOR_PARAMETRO**: Essa tabela armazena os valores aceitos por um determinado parâmetro de uma regra de acesso. Por exemplo, os seguintes valores poderiam estar cadastrados: ‘América’; ‘Ásia’. Além disso, os valores cadastrados nessa tabela devem indicar no atributo *VPR_EXPRESSAO* se ele é ou não uma expressão (cadastrando ' S' caso verdadeiro e ' N' caso falso).
- **FUNCAO**: Essa tabela armazena as funções de autorização de acesso criadas para aplicar as políticas do VPD. Ela possui um relacionamento com a tabela *ESQUEMA* que indica o esquema no qual a função foi definida.

- **ESQUEMA**: Essa tabela armazena os esquemas das tabelas que possuem controle de acesso, além dos esquemas das funções de autorização de acesso das políticas do VPD.
- **APLICACAO_POLITICA**: Essa tabela armazena os nomes das políticas do VPD aplicadas aos conceitos. Cada conceito pode possuir várias políticas e cada uma dessas políticas pode ser associada a uma função diferente. Para representar isso a tabela *APLICACAO_POLITICA* define um relacionamento M × N entre as tabelas *CONCEITO* e *FUNCAO*.
    - Como mencionado anteriormente, uma política possui a propriedade opcional de mascarar as colunas relevantes. Para definir a utilização ou não dessa propriedade pelo VPD, foi criado o campo *APL_MASCARAR*, que deve receber o valor 'S' quando a propriedade for utilizada pela política e o valor 'N' quando não for utilizada.
    - Outra propriedade que uma política pode utilizar é o *UPDATE_CHECK*. Essa propriedade recebeu a mesma abordagem da anterior. Nesse caso foi criado o campo *APL_UPDATE_CHECK*, que deve receber o valor 'S' quando a propriedade for utilizada pela política e o valor 'N' quando não for utilizada. Essa é a propriedade que define se uma política deve ser aplicada ao resultado de um *Insert* ou de um *Update*. A propriedade *UPDATE_CHECK* será aprofundada nos próximos capítulos.
- **APLICACAO_POLITICA_ATRIBUTO**: Essa tabela cria o relacionamento M × N entre a política e os atributos do conceito, definindo as colunas relevantes para o controle de acesso. Se nenhuma coluna for indicada como relevante a política será aplicada á tabela inteira.

4.3 Função genérica de política do VPD

Para aplicar as políticas de segurança através do VPD, foi implementada uma função genérica responsável por acessar as tabelas do modelo de armazenamento de regras de autorização e construir o predicado VPD da política. A seguir é apresentada a função genérica (Figura 8) e descritos os passos realizados pela função.

```
create or replace
FUNCTION "F_POLICY" (p_schema in varchar2, p_tab in varchar2) return varchar2
as
  -- Armazena o usuário do sistema que está executando a consulta
v_user varchar2 (100);

  -- Armazena o predicado dos perfis do usuário
v_return_string varchar2 (10000) default '';

  -- Armazena o número de perfis ativos no conceito
v_numero_perfis_ativos number default 0;

  -- Armazena a regra do perfil do usuário (parte do predicado sem parâmetros
  -- específicos que é associada aos perfis pais e aos perfis sem hierarquia)
v_regra FARBAC.REGRA.RGR_SQL%type;

  -- Armazena a ID associada à regra do perfil do usuário
v_regra_id FARBAC.REGRA.RGR_ID%type;

  -- Armazena a ID associada ao perfil do usuário
v_perfil_id FARBAC.PERFIL.PRF_ID%type;

  -- Recupera as regras dos perfis do usuário, as IDs associadas a essas regras
  -- e as IDs associadas a esses perfis
cursor cur_regras_perfil
is
  select RGR.RGR_SQL, RGR.RGR_ID, PRF_FILHO.PRF_ID
  from FARBAC.REGRA RGR
  inner join FARBAC.REGRA_CONCEITO RC on RC.RC_RGR_ID = RGR.RGR_ID
  inner join FARBAC.CONCEITO CNT on CNT.CNT_ID = RC.RC_CNT_ID
  inner join FARBAC.PERFIL_REGRA PR on PR.PR_RGR_ID = RGR.RGR_ID
  inner join FARBAC.PERFIL PRF_PAI on PRF_PAI.PRF_ID = PR.PR_PRF_ID
  inner join FARBAC.PERFIL PRF_FILHO on PRF_FILHO.PRF_PAI_ID = PRF_PAI.PRF_ID
  inner join FARBAC.PERFIL_USUARIO PU on PU.PU_PRF_ID = PRF_FILHO.PRF_ID
  inner join FARBAC.USUARIO USR on PU.PU_USR_CHAVE = USR.USR_CHAVE
  where LOWER(USR.USR_CHAVE) = LOWER(v_user)
    and LOWER(CNT.CNT_NOME) = LOWER(p_tab)
    and (PRF_FILHO.PRF_DATA_EXPIRACAO > sysdate
         or PRF_FILHO.PRF_DATA_EXPIRACAO is null)
    and (PU.PU_DATA_EXPIRACAO > sysdate or PU.PU_DATA_EXPIRACAO is null)
    and (PRF_PAI.PRF_DATA_EXPIRACAO > sysdate
         or PRF_PAI.PRF_DATA_EXPIRACAO is null);

  -- Armazena os parâmetros do perfil do usuário (parte do predicado com
  -- parâmetros específicos)
v_parametros varchar2 (10000) default '';

  -- Armazena o nome do conceito que possui um campo que é um atributo que faz
  -- parte de um parâmetro do perfil do usuário
v_par_conceito_nome FARBAC.CONCEITO.CNT_NOME%type;

  -- Armazena o atributo que faz parte de um parâmetro do perfil do usuário
v_par_atributo_nome FARBAC.ATRIBUTO.ATR_NOME%type;

  -- Armazena o tipo do atributo que faz parte de um parâmetro do perfil
  -- do usuário
v_par_atributo_tipo FARBAC.ATRIBUTO.ATR_DATA_TYPE %type;

  -- Armazena o símbolo de um operador que faz parte de um parâmetro do perfil
  -- do usuário
```

```
30 v_par_operador_simbolo FARBAC.OPERADOR.OPR_SIMBOLO%type;

   -- Recupera os atributos com seus tipos e seus conceitos de origem e os
   -- operadores que formam os parâmetros da regra do perfil do usuário
31 cursor cur_parametros_regra
32 is
33   select CNT.CNT_NOME, ATR.ATR_NOME, ATR.ATR_DATA_TYPE, OPR.OPR_SIMBOLO
34   from FARBAC.REGRA RGR
35   inner join FARBAC.PARAMETRO PAR on PAR.PAR_RGR_ID = RGR.RGR_ID
36   inner join FARBAC.ATRIBUTO ATR on ATR.ATR_ID = PAR.PAR_ATR_ID
37   inner join FARBAC.OPERADOR OPR on OPR.OPR_ID = PAR.PAR_OPR_ID
38   inner join FARBAC.CONCEITO CNT on ATR.ATR_CNT_ID = CNT.CNT_ID
39   where RGR.RGR_ID = v_regra_id;

   -- Armazena todos os valores aceitos por um parâmetro do perfil do usuário
40 v_valores_parametro varchar2 (10000) default '';

   -- Armazena um dos valores aceitos por um parâmetro do perfil do usuário
41 v_valor_parametro FARBAC.VALOR_PARAMETRO.VPR_VALOR%type;

   -- Armazena o indicador se o valor do parâmetro é uma expressão
42 v_valor_expressao FARBAC.VALOR_PARAMETRO.VPR_EXPRESSAO%type;

   -- Recupera os valores aceitos pelos parâmetros do perfil do usuário,
   -- e o indicador do valor como expressão
43 cursor cur_valores_parametro
44 is
45   select VPR.VPR_VALOR, VPR.VPR_EXPRESSAO
46   from FARBAC.VALOR_PARAMETRO VPR
47   inner join FARBAC.PARAMETRO PAR on PAR.PAR_ID = VPR.VPR_PAR_ID
48   inner join FARBAC.ATRIBUTO ATR on ATR.ATR_ID = PAR.PAR_ATR_ID
49   inner join FARBAC.CONCEITO CNT on CNT.CNT_ID = ATR.ATR_CNT_ID
50   where LOWER(CNT.CNT_NOME) = LOWER(v_par_conceito_nome)
51     and VPR.VPR_PRF_ID = v_perfil_id
52     and LOWER(ATR.ATR_NOME) =  LOWER(v_par_atributo_nome);

53 begin
     -- Recupera usuário que está realizando a consulta
54   v_user := lower(sys_context('userenv', 'session_user'));
55   if v_user != 'secuser' then
56     null;
57   else
58     v_user := lower(sys_context('userenv','proxy_user'));
59   end if;

     -- Abre o predicado
60   v_return_string := '1=2 OR (';

     -- Abre o cursor de regras e recupera as regras de todos os perfis do usuário
61   open cur_regras_perfil;
62   loop
63     fetch cur_regras_perfil into v_regra, v_regra_id, v_perfil_id ;
64     exit when cur_regras_perfil%notfound;

       -- Abre o cursor de parâmetros e recupera todos os parâmetros da regra do
       -- perfil do usuário
       -- Está em loop aninhado, então faz isso para todas as regras dos perfis
       -- do usuário
65     open cur_parametros_regra;
       -- Limpa os parâmetros concatenados na última abertura do cursor
```

```
66      v_parametros := '';
67      loop
68        fetch cur_parametros_regra into v_par_conceito_nome, v_par_atributo_nome,
                                         v_par_atributo_tipo, v_par_operador_simbolo;
69        exit when cur_parametros_regra%notfound;

          -- Concatena os parâmetros com 'AND' entre si
70        if length(v_parametros) is not null then
71          v_parametros := v_parametros || ' AND ';
72        end if;

          -- Abre o cursor de valores do parâmetro e recupera todos os valores
          -- aceitos pelo parâmetro do perfil do usuário
          -- Está em loop aninhado, então faz isso para todos os parâmetros do
          -- perfil do usuário
73        open cur_valores_parametro;
          -- Limpa os valores dos parâmetros concatenados na última abertura
          -- do cursor
74        v_valores_parametro := '(';
75        loop
76          fetch cur_valores_parametro into v_valor_parametro, v_valor_expressao;
77          exit when cur_valores_parametro%notfound;

            -- Concatena os valores dos parâmetros com ',' entre si
78          if v_valores_parametro  <> '(' then
79            v_valores_parametro := v_valores_parametro  || ' , ';
80          end if;

            -- Se o valor do parâmetro não for uma expressão ou do tipo NUMBER,
            -- o valor deve ser concatenado entre '
81          if lower(v_valor_expressao) = 's'
                    or lower(v_par_atributo_tipo) = 'number' then
82            v_valores_parametro := v_valores_parametro  || v_valor_parametro;
83          else
84            v_valores_parametro := v_valores_parametro  || '''' ||
                                     v_valor_parametro || '''';
85          end if;
86        end loop;
87        close cur_valores_parametro;

          -- Fecha os valores do parâmetro
88        v_valores_parametro := v_valores_parametro || ')';

          -- Monta os parâmetros do perfil
89        v_parametros := v_parametros || v_par_atributo_nome || ' ' ||
                          v_par_operador_simbolo || ' ' || v_valores_parametro;

90      end loop;
91      close cur_parametros_regra;

        -- Concatena os predicados dos perfis com 'OR' entre si (regras permissivas)
92      if v_return_string <> '1=2 OR (' then
93        v_return_string := v_return_string || ' OR ';
94      end if;

        -- Monta o predicado do perfil
        -- Se existir uma regra para o pai, o parênteses deve ser fechado no fim
        -- do predicado
95      if length(v_regra) is not null then
96        v_return_string :=  v_return_string || v_regra || v_parametros || ')';
```

```
 97     else
 98       v_return_string :=  v_return_string || v_parametros;
 99     end if;

100   end loop;
101   close cur_regras_perfil;

      -- Fecha o predicado
102   v_return_string := v_return_string || ')';

      -- Substitui o coringa ?user pelo usuário que está executando a consulta.
103   v_return_string := replace(v_return_string, '?user', v_user);

      -- Verifica se o usuário não possui nenhum perfil
104   if v_return_string = '1=2 OR ()'  then
        -- Recupera o número de perfis ativos associados ao conceito onde o comando
        -- foi executado
105     select count(*) into v_numero_perfis_ativos
106     from FARBAC.PERFIL PRF
107     inner join FARBAC.PERFIL_REGRA PR on PR.PR_PRF_ID = PRF.PRF_ID
108     inner join FARBAC.REGRA RGR on RGR.RGR_ID = PR.PR_RGR_ID
109     inner join FARBAC.REGRA_CONCEITO RC on RC.RC_RGR_ID = RGR.RGR_ID
110     inner join FARBAC.CONCEITO CNT on CNT.CNT_ID = RC.RC_CNT_ID
111     where LOWER(CNT.CNT_NOME) = LOWER(p_tab)
112       and (PRF.PRF_DATA_EXPIRACAO > sysdate or PRF.PRF_DATA_EXPIRACAO is null);

      -- Se o conceito possuir algum perfil ativo associado à ela, retorna '1=2'
      -- senão retorna  '1=1'
113   if (v_numero_perfis_ativos > 0) then
114       v_return_string := '1=2';
115   else
116       v_return_string := '1=1';
117   end if;
118   end if;

119   return v_return_string;
120 end;
```

Figura 8 – Função genérica de política do VPD

O primeiro passo executado pela função genérica é a captura do usuário que está executando a consulta. Em seguida, para cada perfil do usuário, é montado o predicado, unindo as informações cadastradas nas tabelas do modelo de armazenamento das regras de autorização. Caso o perfil do usuário não possua um predicado específico para aquele conceito, deve-se checar se o conceito tem alguma outra regra de autorização válida, o que significa que o conceito tem acesso controlado. Caso o conceito tenha uma regra ainda válida então é retornado '1=2', o que bloqueia o acesso a todo o conteúdo do conceito. Não possuindo nenhuma regra válida, então é retornado '1=1', liberando o acesso ao conceito.

Por padrão toda função de política VPD recebe como parâmetros o nome do esquema e o nome do objeto que disparou a aplicação da política. No primeiro bloco

da função, compreendido entre as linhas 4 e 52, encontram-se as declarações das variáveis e dos cursores. Cada variável e cada cursor possui um comentário com uma breve descrição, contudo é interessante detalharmos um pouco mais os cursores.

O primeiro cursor chamado de *cur_regras_perfil* recupera as regras e os perfis do usuário (linhas 10 a 25). A consulta do cursor parte da tabela *REGRA* (linha 13) e então é feita uma junção com a tabela *REGRA_CONCEITO* (linha 14) e outra junção com a tabela *CONCEITO* (linha 15), dessa forma obtendo os perfis que estão associados ao CONCEITO cuja regra de autorização está sendo aplicada (explicitada no parâmetro *LOWER(CNT.CNT_NOME) = LOWER(p_tab)* – linha 22). A junção com a tabela *PERFIL_REGRA* (linha 16) e com a tabela *PERFIL* (linha 17) permite fazer a relação Perfil × Conceito. Contudo, a tabela *REGRA* recebe uma entrada para o perfil pai da hierarquia, enquanto os usuários são associados a um perfil filho, desse modo, é necessário montar a hierarquia para podermos achar o perfil pai do perfil do usuário. Para isso é feita mais uma junção com a tabela *PERFIL* (linha 18), formando a hierarquia, e uma junção com a tabela *USUARIO*, para que possamos capturar apenas o perfil do usuário que está realizando a consulta (linhas 19 e 20). A última parte da consulta são os parâmetros. O primeiro especifica o usuário que está realizando a consulta (linha 21), o segundo especifica o conceito onde a consulta é realizada (linha 22), o terceiro verifica se o perfil ainda está válido (linha 23), o quarto verifica se a aplicação do perfil ao usuário ainda está válida (linha 24) e finalmente, o quinto verifica se o perfil pai ainda está válido (linha 25).

O segundo cursor, chamado *cur_parametros_regra*, captura os componentes dos parâmetros do perfil pai do perfil do usuário (linhas 31 a 39). Lembrando que os parâmetros do perfil são associados à regra cadastrada da tabela *REGRA* para o perfil pai; portanto, essa tabela deve estar incluída na consulta. A consulta do cursor parte da tabela *REGRA* (linha 34) e então faz uma junção com a tabela *PARAMETRO* (linha 35), capturando todos os parâmetros para aquela *REGRA*. Como o parâmetro é composto por atributo, operador e valores, devem ser feitas mais três junções, uma com a tabela *ATRIBUTO* (linha 36), outra com a tabela *OPERADOR* (linha 37) e mais uma com a tabela *CONCEITO* (linha 38). Essa última junção é necessária para obtermos o nome do conceito do atributo do parâmetro, que será usado mais tarde para recuperar dinamicamente os valores aceitos pelo parâmetro. Finalmente a

consulta é encerrada com um parâmetro que especifica de qual *REGRA* são os parâmetros que desejamos recuperar (linha 39).

O terceiro cursor, chamado *cur_valores_parametro*, captura os valores aceitos para os parâmetros do perfil do usuário (linhas 43 a 52). Estes valores são utilizados para completar o predicado VPD da política. A consulta do cursor parte da tabela *VALOR_PARAMETRO* (linha 46). Em seguida, faz junção com a tabela *PARAMETRO* (linha 47) a fim de recuperar os valores do parâmetro correto. Em seguida, descobre de qual atributo e de qual tabela está se buscando o valor, através das junções com as tabelas *ATRIBUTO* e *CONCEITO* (linhas 48 e 49). Para tal, é necessário indicar qual é o conceito (linha 50), qual é o nome do atributo (linha 52) e de qual perfil está buscando o valor (linha 51).

O segundo bloco da função, compreendido entre as linhas 53 e 120, monta o predicado da política propriamente dito. O primeiro passo é capturar o usuário que realizou a consulta, isso é feito acessando o contexto *userenv* através da função *sys_context* da *Application Context* (linhas 54 a 59). Em seguida o predicado é iniciado com '1=2 OR (' (linha 60), o cursor *cur_regras_perfil* é aberto (linha 61), e entra em um loop onde será montado o predicado (linha 62). O loop percorrerá todas as regras de perfil cadastradas para esse usuário na tabela *REGRA*, mas só haverá mais de um se o usuário tiver mais de um perfil com regra de autorização para o conceito sobre a qual a política está sendo aplicada. Dentro do loop é aberto o cursor *cur_parametros_perfil* (linha 65). Em seguida, entra em outro loop para recuperar todos os parâmetros do perfil (linha 67). O loop é aninhado porque, caso o usuário tenha mais de uma regra de perfil, os parâmetros terão que ser montados para cada uma delas, portanto, antes de entrar no segundo loop, é necessário também limpar a string que armazena os parâmetros montados (linha 66).

Dentro do segundo loop a string com os parâmetros é montada, onde os parâmetros são unidos entre si por AND (linha 72). Cada parâmetro é formado da concatenação do atributo com o operador e com os valores aceitos pelo parâmetro. Os valores para os parâmetros são obtidos através do cursor *cur_valores_parametro* (linha 73). Um loop é executado neste cursor (linha 75), concatenando os valores separados por vírgula (linha 79) e verificando se o tipo de dado do valor corresponde

a uma expressão, número ou qualquer outro tipo (linhas 81 a 85). Observe que caso o valor não seja nem uma expressão nem um número (ex.: VARCHAR2), deve ser posto entre aspas simples (linha 84). Ao sair do segundo loop, o predicado do perfil é montado concatenando a regra do perfil com os parâmetros do perfil (linha 96), no caso de existir uma regra de perfil, ou concatenando apenas os parâmetros do perfil (linha 98), no caso de não existir uma regra de perfil. Observe que as regras dos perfis são concatenadas por OR (linhas 92 a 94) caso o usuário possua mais de um perfil para o conceito.

Caso seja necessário utilizar o identificador do usuário na definição regra, pode-se usar o coringa '?user', pois a função faz a substituição deste valor pelo identificador do usuário (linha 103) recuperado através do *Application Context*.

Finalmente, se o usuário não possuir nenhum perfil com regra de autorização para esse conceito (linha 104), será feita uma consulta para verificar se existe algum perfil ativo que possua regra de acesso para o conceito (linhas 105 a 112). Se tal perfil existir, será retornado o predicado '1=2', impedindo o acesso aos dados do conceito, caso contrário será retornado o predicado '1=1', liberando o acesso aos dados.

4.4 Função de geração do comando de inserção/remoção de política do VPD

A aplicação e a remoção de políticas VPD são feitas com a API *DBMS_RLS* do Oracle, através dos comandos apresentado nas Figura 9 e Figura 10 respectivamente. O parâmetro *sec_relevant_cols* é opcional e define as colunas relevantes para a política de segurança, ou seja, o controle só será efetuado para as consultas que envolverem tais colunas. Se nenhuma coluna for definida, o controle será válido para todas as colunas do conceito. O parâmetro *sec_relevant_cols_opt* depende do parâmetro anterior, e é o que define se os dados sensíveis das colunas relevantes serão mascarados ou não. Se o valor *DBMS_RLS.ALL_ROWS* for definido a opção de mascaramento será ativada, por padrão ela é desativada. O parâmetro *update_check* define o comportamento do VPD em inserções e atualizações. Caso ele seja definido como *true*, o resultado das inserções e das atualizações será checado de

acordo com a política definida. Caso ele seja definido como *false*, os dados inseridos e atualizados não serão checados de acordo com a política definida, de modo que o usuário poderá inserir/atualizar um dado para um estado que ele não terá acesso. O valor padrão do parâmetro *update_check* é *false*.

```
DBMS_RLS.ADD_POLICY ('esquema do objeto',
                     'nome do objeto',
                     'nome da política',
                     'esquema da função',
                     'nome da função',
                     sec_relevant_cols => 'colunas relevantes',
                     sec_relevant_cols_opt => DBMS_RLS.ALL_ROWS
                     update_check => boolean);
```

Figura 9 – Sintaxe do comando de aplicação de política do VPD

```
DBMS_RLS.DROP_POLICY ('esquema do objeto',
                      'nome do objeto',
                      'nome da política');
```

Figura 10 – Sintaxe do comando de remoção de política do VPD

A princípio, estes comandos deveriam ser digitados pela equipe responsável pelo cadastro e remoção das políticas do VPD. Entretanto, o modelo de armazenamento de regras de autorização já armazena as informações necessárias para montar o comando dinamicamente, portanto, utilizando a função apresentada na Figura 11, é possível gerar o comando para criação da política (com o parâmetro *p_tipo_comando = 'C'*) e para remoção da política (com o parâmetro *p_tipo_comando = 'R'*).

```
 1 create or replace
 2 FUNCTION "F_COMANDO_POLITICA" (p_nome_politica in varchar2,
                                  p_tipo_comando in char) return varchar2
 3 as
   -- Armazena o comando de criação/remoção da política que será retornado
   -- pela função
 4 v_comando varchar2 (1000) default'';

   -- Armazena o nome do esquema do conceito
 5 v_conceito_esquema FARBAC.ESQUEMA.ESQ_NOME%type;

   -- Armazena o nome do conceito
 6 v_conceito_nome FARBAC.CONCEITO.CNT_NOME%type;

   -- Armazena o nome do esquema da função
 7 v_funcao_esquema FARBAC.ESQUEMA.ESQ_NOME%type;

   -- Armazena o nome da função
 8 v_funcao_nome FARBAC.FUNCAO.FUN_NOME%type;

   -- Armazena o nome de uma coluna relevante da política
```

```
 9 v_coluna_relevante FARBAC.ATRIBUTO.ATR_NOME%type;

   -- Armazena o parâmetro de colunas relevantes da política
10 v_parametro_colunas_relevantes varchar2 (500) default'';

   -- Armazena o indicador da opção de mascaramento da política
11 v_mascarar FARBAC.APLICACAO_POLITICA.APL_MASCARAR%type;

   -- Armazena o parâmetro de mascaramento da política
12 v_parametro_mascarar varchar2 (45) default'';

   -- Armazena o indicador da opção de update check da política
13 v_update_check FARBAC.APLICACAO_POLITICA.APL_UPDATE_CHECK%type;

   -- Armazena o parâmetro de update check da política
14 v_parametro_update_check varchar2 (25) default'';

   -- Recupera as colunas relevantes da política
15 cursor cur_colunas_relevantes
16 is
17   select ATR.ATR_NOME
18   from FARBAC.ATRIBUTO ATR
19   inner join FARBAC.APLICACAO_POLITICA_ATRIBUTO APA
            on APA.APA_ATR_ID = ATR.ATR_ID
20   inner join FARBAC.APLICACAO_POLITICA APL on APL.APL_ID = APA.APA_APL_ID
21   where LOWER(APL.APL_NOME_POLITICA) = LOWER(p_nome_politica);

22 begin
     -- Captura os valores dos parâmetros do comando de criação/remoção da política
23   select APL.APL_MASCARAR, APL.APL_UPDATE_CHECK, ESQ_FUN.ESQ_NOME, FUN.FUN_NOME,
           ESQ_CNT.ESQ_NOME, CNT.CNT_NOME
     into v_mascarar, v_update_check, v_funcao_esquema, v_funcao_nome,
          v_conceito_esquema, v_conceito_nome
24   from FARBAC.APLICACAO_POLITICA APL
25   inner join FARBAC.FUNCAO FUN on FUN.FUN_ID = APL.APL_FUNC_ID
26   inner join FARBAC.CONCEITO CNT on CNT.CNT_ID = APL.APL_CNT_ID
27   inner join FARBAC.ESQUEMA ESQ_FUN on ESQ_FUN.ESQ_ID = FUN.FUN_ESQ_ID
28   inner join FARBAC.ESQUEMA ESQ_CNT on ESQ_CNT.ESQ_ID = CNT.CNT_ESQ_ID
29   where LOWER(APL.APL_NOME_POLITICA) = LOWER(p_nome_politica);

     -- Se a opção de update check for marcada como "Sim", define o valor do
     -- parâmetro como "true"
     -- Se a opção de update check for marcada como "Não", define o valor do
     -- parâmetro como "false"
30   if lower(v_update_check) = 's' then
31     v_parametro_update_check := ', update_check => true';
32   else
33     v_parametro_update_check := ', update_check => false';
34   end if;

     -- Se a opção de mascarar colunas relevantes for marcada como "Sim", define
     -- o parâmetro extra "sec_relevant_cols_opt => dbms_rls.ALL_ROWS"
     -- Se a opção de mascarar colunas relevantes for marcada como "Não", não define
     -- nenhum parâmetro extra
35   if lower(v_mascarar) = 's' then
36     v_parametro_mascarar := ', sec_relevant_cols_opt => DBMS_RLS.ALL_ROWS';
37   else
38     v_parametro_mascarar := '';
39   end if;
```

```
    -- Monta a string com os nomes das colunas relevantes
40  open cur_colunas_relevantes;
41  loop
42    fetch cur_colunas_relevantes into v_coluna_relevante;
43    exit when cur_colunas_relevantes%notfound;

44    if length(v_parametro_colunas_relevantes) is not null then
45        v_parametro_colunas_relevantes := v_parametro_colunas_relevantes || ', ';
46    end if;

47    v_parametro_colunas_relevantes := v_parametro_colunas_relevantes ||
                                        v_coluna_relevante;

48  end loop;
49  close cur_colunas_relevantes;

    -- Se a política possuir alguma coluna relevantes, especificar o parâmetro
    -- "sec_relevant_cols =>" antes dos nomes das colunas
50  if length(v_parametro_colunas_relevantes) is not null then
51        v_parametro_colunas_relevantes := ', sec_relevant_cols => ''' ||
                                            v_parametro_colunas_relevantes || '''';
52  end if;

    -- Monta o comando.
      -- 'c' - Criação
      -- 'r' - Remoção
53  if lower(p_tipo_comando) = 'c' then
54    v_comando := 'dbms_rls.add_policy('
                || 'object_schema => ''' || v_conceito_esquema
                || ''', object_name => ''' || v_conceito_nome
                || ''', policy_name => ''' || p_nome_politica
                || ''', function_schema => ''' || v_funcao_esquema
                || ''', policy_function => ''' || v_funcao_nome || ''''
                || v_parametro_update_check
                || v_parametro_colunas_relevantes
                || v_parametro_mascarar || ');';
55  else if lower(p_tipo_comando) = 'r' then
56    v_comando := 'dbms_rls.drop_policy('
                || 'object_schema => ''' || v_conceito_esquema
                || ''', object_name => ''' || v_conceito_nome
                || ''', policy_name => ''' || p_nome_politica || ''');';
57  end if;
58  end if;

59  return v_comando;
60 end;
```

Figura 11 – Função de geração de comando de inserção/remoção de política VPD

O primeiro passo executado pela função de geração de comando para criação/remoção de política VPD, é capturar as informações que definem a política de acordo com o nome da política. Em seguida, de posse dessas informações, os parâmetros básicos do comando são montados. Se a política é aplicada a colunas relevantes, essas colunas são capturadas e o parâmetro de colunas relevantes é montado. Por fim, dependendo do tipo de comando desejado (criação ou remoção), o

método correto da API *DBMS_RLS* é definido e concatenado com os parâmetros gerados anteriormente.

No primeiro bloco da função, compreendido entre as linhas 4 e 21, encontram-se as declarações das variáveis e do cursor. Cada variável e cursor possuem um comentário com uma breve descrição. O cursor *cur_colunas_relevantes* recupera as colunas relevantes para a política (linhas 15 a 21). A consulta do cursor parte da tabela *ATRIBUTO* (linha 18) e então é feita uma junção com a tabela *APLICACAO_POLITICA_ATRIBUTO* (linha 19) e outra com a tabela *APLICACAO_POLITICA* (linha 20), dessa forma obtendo as colunas (atributos) relevantes à política. O parâmetro da consulta define a política que está sendo tratada (linha 21).

O segundo bloco da função, compreendido entre as linhas 22 e 60, monta o comando propriamente dito. O primeiro passo é recuperar as opções da política, isso é feito através da consulta compreendida entre as linhas 23 e 29. A consulta parte da tabela *APLICACAO_POLITICA* (linha 24) e faz uma junção com a tabela *FUNCAO* (linha 25), com a tabela *CONCEITO* (linha 26) e duas junções com a tabela *ESQUEMA* (linhas 27 e 28), uma referente ao esquema da função e outra referente ao esquema do conceito. Essas junções capturam as opções do comando que são: nome e esquema do conceito, nome e esquema da função e opções de mascaramento e de *update_check*. O parâmetro da consulta define a política que está sendo tratada (linha 29).

Em seguida, de acordo com valor definido para a opção *update_check*, o parâmetro homônimo é definido como *true* ou como *false* (linhas 30 a 34). Analogamente, de acordo com a opção defina para mascaramento, o parâmetro *sec_relevant_cols_opt => DBMS_RLS.ALL_ROWS* é definido ou não (linhas 35 a 39).

No próximo passo o cursor *cur_colunas_relevantes* é aberto (linha 40), e entra em um loop onde será montada a lista de colunas relevantes (linha 41). O loop percorrerá todas as colunas relevantes para a política cadastradas na tabela *APLICACAO_POLITICA_ATRIBUTO*, e as concatenará separadas por vírgula (linha 45). Ao sair do loop, se alguma coluna relevante foi encontrada para a política, o

parâmetro *sec_relevant_cols* => é definido recebendo o nome dessas colunas (linhas 50 a 52).

Por fim, de acordo com o tipo de comando desejado, o método correto da API *DBMS_RLS* é selecionado (*add_policy* para criação e *drop_policy* para remoção) e concatenado aos parâmetros definidos para a política (linhas 53 a 58). Para melhorar o entendimento do funcionamento da função, a seguir são apresentados dois exemplos de uso da função para retornar o comando de criação e o comando de remoção de política VPD.

No primeiro exemplo foi feito um registro na tabela *APLICACAO_POLITICA* com o nome da política definido como *ACCESS_POLICY_T1*. A função utilizada pela política e cadastrada na tabela *FUNCAO* chama-se *F_POLICY*, e foi associada ao esquema *FARBAC*, cadastrado na tabela *ESQUEMA*. A tabela onde a política se aplica e cadastrada na tabela *CONCEITO* chama-se *EMPREGADOS*, e foi associada ao esquema *RH*, cadastrado na tabela *ESQUEMA*.

A propriedade *MASCARAR* foi definida como 'N' enquanto a propriedade *UPDATE_CHECK* foi definida como 'S'. Nenhuma coluna relevante foi associada à política.

A chamada da função para retornar o comando de criação e o comando de criação retornado para a política *ACCESS_POLICY_T1* são apresentados na Figura 12, enquanto a chamada da função para retornar o comando de remoção e o comando de remoção retornado para a política *ACCESS_POLICY_T1* são apresentados na Figura 13.

```
//Consulta para retornar o comando de criação
select f_comando_politica ('ACCESS_POLICY_T1', 'C') from dual;

//Comando retornado
dbms_rls.add_policy(
  object_schema => RH,
  object_name => 'EMPREGADOS',
  policy_name => 'ACCESS_POLICY_T1',
  function_schema => 'FARBAC',
  policy_function => 'F_POLICY',
  update_check => true);
```

Figura 12 - Chamada da função de retorno do comando de criação e comando de criação para a política *ACCESS_POLICY_T1*

```
//Consulta para retornar o comando de remoção
select f_comando_politica ('ACCESS_POLICY_T1', 'R') from dual;

//Comando retornado
dbms_rls.drop_policy(
  object_schema => 'RH',
  object_name => 'EMPREGADOS',
  policy_name => 'ACCESS_POLICY_T1');
```

Figura 13 - Chamada da função de retorno do comando de remoção e comando de remoção para a política *ACCESS_POLICY_T1*

No segundo exemplo foi feito um registro na tabela *APLICACAO_POLITICA* com o nome da política definido como *ACCESS_POLICY_T2*. A função utilizada pela política e cadastrada na tabela *FUNCAO* chama-se *F_POLICY_COLUMN*, e foi associada ao esquema *FARBAC*, cadastrado na tabela *ESQUEMA*. A tabela para essa política é a mesma do exemplo anterior (*EMPREGADOS*).

A propriedade *UPDATE_CHECK* foi definida como 'S', assim como a propriedade *MASCARAR*. Por fim os atributos *MATRICULA*, *CPF* e *RG* cadastrados na tabela *ATRIBUTO* foram definidos como colunas relevantes, através do relacionamento criado na tabela *APLICACAO_POLITICA_ATRIBUTO*.

A chamada da função para retornar o comando de criação e o comando de criação retornado para a política *ACCESS_POLICY_T2* são apresentados na Figura 14, enquanto a chamada da função para retornar o comando de remoção e o comando de remoção retornado para a política *ACCESS_POLICY_T2* são apresentados Figura 15.

```
//Consulta para retornar o comando de criação
select f_comando_politica ('ACCESS_POLICY_T2', 'C') from dual;

//Comando retornado
dbms_rls.add_policy(
  object_schema => 'RH',
  object_name => 'EMPREGADOS',
  policy_name => 'ACCESS_POLICY_T2',
  function_schema => 'FARBAC',
  policy_function => 'F_POLICY_COLUMN',
  update_check => true,
  sec_relevant_cols => 'MATRICULA, CPF, RG',
  sec_relevant_cols_opt => DBMS_RLS.ALL_ROWS);
```

Figura 14 - Chamada da função de retorno do comando de criação e comando de criação para a política *ACCESS_POLICY_T2*

```
//Consulta para retornar o comando de remoção
select f_comando_politica ('ACCESS_POLICY_T2', 'R') from dual;

//Comando retornado
dbms_rls.drop_policy(
  object_schema => 'RH',
  object_name => 'EMPREGADOS',
  policy_name => 'ACCESS_POLICY_T2');
```

Figura 15 - Chamada da função de retorno do comando de remoção e comando de remoção para a política *ACCESS_POLICY_T2*

Capítulo 5: AVALIAÇÃO EXPERIMENTAL

Para avaliar a capacidade do *framework* proposto, um conjunto de regras de acesso foi definido sob o esquema do *benchmark* TPC-H, e uma série de testes experimentais contemplando várias operações SQL foram executados a fim de avaliar o comportamento do SGBD. Esse capítulo apresenta uma descrição do TPC-H, a listagem de regras de acesso definidas bem como o resultado dos testes realizados com as regras.

5.1 *Benchmark* TPC-H

O TPC-H é uma especificação de *benchmark* com uma ampla relevância na indústria, que simula o cenário de uma aplicação de apoio à decisão [TPC *Council*, 2008]. Esse cenário possui um contexto realista que representa as atividades de uma indústria de atacado que precisa gerenciar as suas vendas ou distribuir um produto em todo o mundo (por exemplo, aluguel de veículos, distribuição de alimentos, etc.). O *benchmark* consiste da descrição de um esquema de banco de dados e de um conjunto de consultas *ad-hoc* OLAP (On-line-Analitical Processing) orientadas ao negócio.

O *benchmark* TPC-H foi escolhido para mostrar a eficiência e generalidade do *framework* proposto, uma vez que analisa um grande volume de dados, executa consultas com um alto grau de complexidade e fornece respostas a questões críticas de negócio.

5.1.1 Modelo de dados do TPC-H

O modelo de dados do TPC-H, considerando apenas os nomes das entidades, é apresentado na Figura 16. Este modelo foi gerado a partir da ferramenta VisualTPCH [Domingues *et al.*, 2008]. As arestas apontam na direção de relacionamentos um para muitos entre tabelas. Logo, uma *NATION* possui vários *CUSTOMERS*. A ferramenta VisualTPCH foi utilizada para gerar automaticamente

todo o esquema do TPC-H e ainda popular as tabelas com dados *dummy*[1] para os testes.

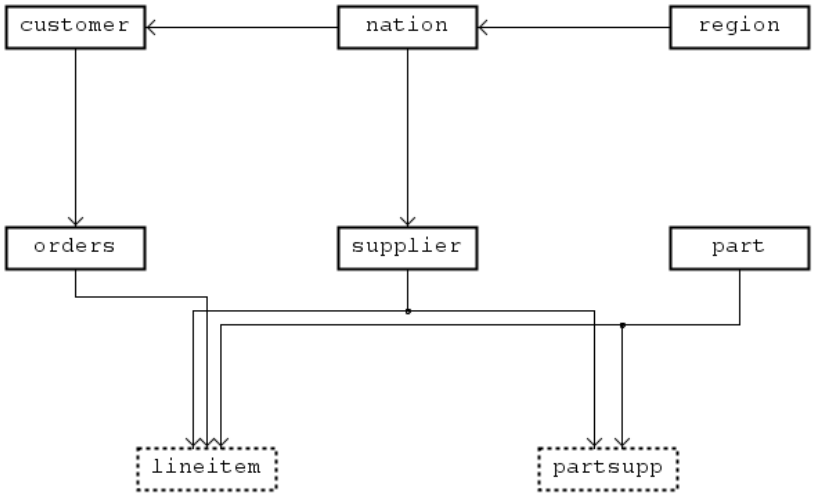

Figura 16 – Modelo de entidades do *benchmark* TPC-H da ferramenta VisualTPCH [Domingues *et al.*, 2008]

A Figura 17 apresenta o modelo do esquema do TPC-H incluindo os atributos das entidades. Os parênteses ao lado do nome da tabela contêm o prefixo dos nomes dos atributos de cada tabela. O número/fórmula abaixo do nome de cada tabela representa a cardinalidade (número de linhas) da tabela. Algumas cardinalidades são definidas com o fator de escala SF (*Scale Factor*), obtendo o tamanho da tabela de acordo com o espaço reservado para os dados. Mais detalhes sobre o modelo podem ser encontrados na especificação do TPC-H [TPC *Council*, 2008].

[1] Chamamos de dados *dummy* os dados que foram cadastrados na base de dados e não fazem sentido para o negócio, servindo apenas para popular a base a fim de termos um volume de dados para testes.

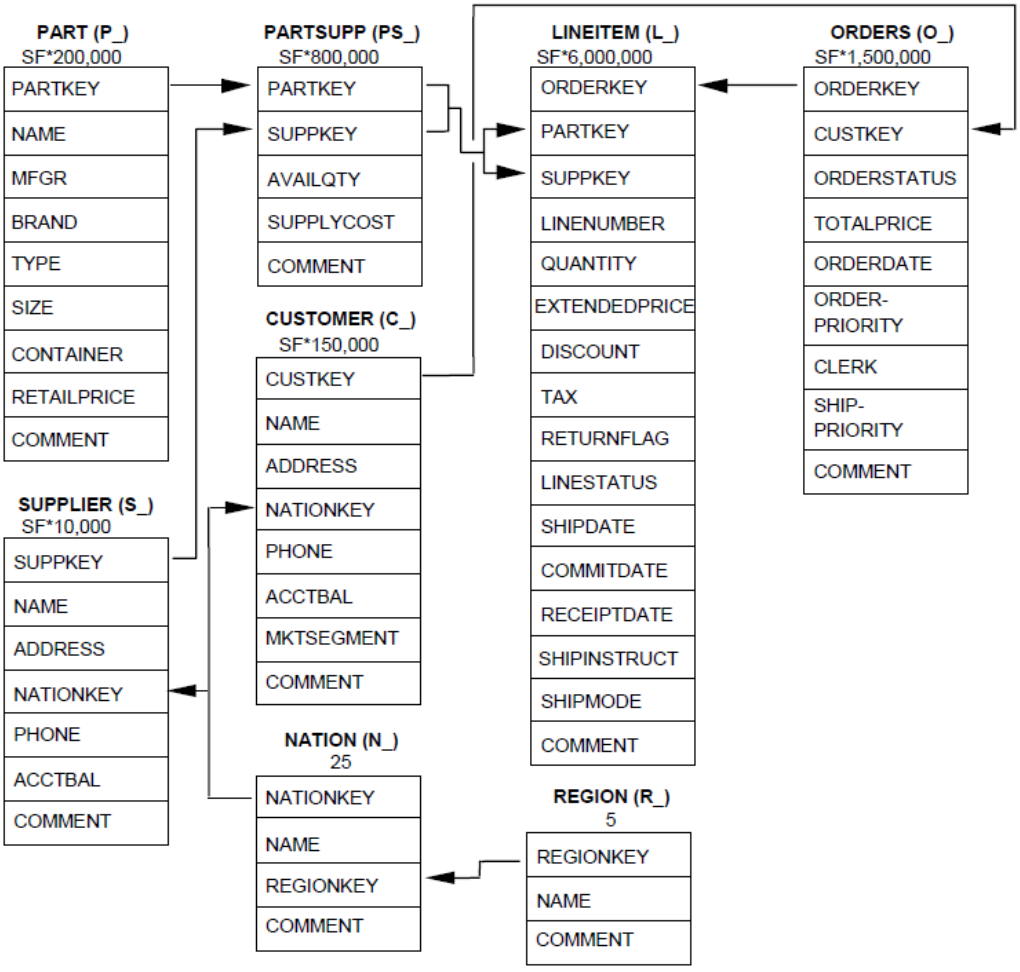

Figura 17 – Modelo do esquema TPC-H [TPC *Council*, 2008]

5.1.2 Consultas do TPC-H

Para a realização dos testes experimentais, algumas consultas do *benchmark* TPC-H foram selecionadas para cobrir padrões comuns de seleção de dados no banco de dados: consultas simples, consultas com sub-consultas (aninhadas), consultas com agrupamento (*group by*), consultas com filtro de agrupamento (*having*) e consultas com *case*. Todas as consultas listadas nessa seção podem ser encontradas na especificação do TPC-H [TPC *Council*, 2008].

C1. **Previsão de mudança na receita**: Quantifica o aumento da receita resultante da eliminação de algum desconto percentual em determinado ano (Figura 18).

```
select sum(l_extendedprice * l_discount) as revenue
from lineitem
where l_shipdate >= date '1994-01-01'
  and l_shipdate < date '1994-01-01' + interval '1' year
  and l_discount between 0.06 - 0.01 and 0.06 + 0.01
  and l_quantity < 24;
```

Figura 18 – Consulta C1: Previsão de mudança na receita

C2. **Fornecedor com menor custo:** Determina o fornecedor que deverá ser selecionado para fornecer uma determinada peça em uma determinada região (Figura 19).

```
select s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_comment
from part, supplier, partsupp, nation, region
where p_partkey = ps_partkey
  and s_suppkey = ps_suppkey
  and p_size = 15
  and p_type like '%BRASS'
  and s_nationkey = n_nationkey
  and n_regionkey = r_regionkey
  and r_name = 'AMERICA'
  and ps_supplycost = (select min(ps_supplycost)
                       from partsupp, supplier, nation, region
                       where p_partkey = ps_partkey
                         and s_suppkey = ps_suppkey
                         and s_nationkey = n_nationkey
                         and n_regionkey = r_regionkey
                         and r_name = 'AMERICA')
order by s_acctbal desc, n_name, s_name, p_partkey;
```

Figura 19 – Consulta C2: Fornecedor com menor custo

C3. **Relatório do resumo de preços:** Lista a quantidade de itens vendidos, remetidos e retornados (Figura 20).

```
select
  l_returnflag,
  l_linestatus,
  sum(l_quantity) as sum_qty,
  sum(l_extendedprice) as sum_base_price,
  sum(l_extendedprice*(1-l_discount)) as sum_disc_price,
  sum(l_extendedprice*(1-l_discount)*(1+l_tax)) as sum_charge,
  avg(l_quantity) as avg_qty,
  avg(l_extendedprice) as avg_price,
  avg(l_discount) as avg_disc,
  count(*) as count_order
from lineitem
where l_shipdate <= date '1998-12-01' - interval '90' day (3)
group by l_returnflag,l_linestatus
order by l_returnflag,l_linestatus;
```

Figura 20 – Consulta C3: Relatório do resumo de preços

C4. **Consumidores de grandes volumes:** Ranqueia os consumidores que realizaram pedidos de grande quantidade, de acordo com o preço total de cada pedido (Figura 21).

```
select c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice, sum(l_quantity)
from customer, orders, lineitem
where o_orderkey in (select l_orderkey
                     from lineitem
                     group by l_orderkey having sum(l_quantity) > 300)
  and c_custkey = o_custkey
  and o_orderkey = l_orderkey
group by c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice
order by o_totalprice desc, o_orderdate;
```

Figura 21 – Consulta C4: Relatório do resumo de preços

A especificação do TPC-H possui uma consulta que determina a mudança na participação de mercado de uma determinada nação para um tipo de peça durante dois anos. Essa consulta foi alterada para determinar a mudança na participação de mercado de uma região inteira, ao invés de uma única nação. A nova consulta é apresentada no item C5.

C5. **Participação de mercado:** Determina a mudança na participação de mercado de uma determinada região para um tipo de peça durante dois anos (Figura 22).

```
select o_year,
   sum(case
       when region = 'AMERICA'
       then volume
       else 0
       end) / sum(volume) as mkt_share
from (select extract(year from o_orderdate) as o_year,
        l_extendedprice * (1-l_discount) as volume, r_name as region
      from part, supplier, lineitem, orders, customer, nation, region
      where p_partkey = l_partkey
        and s_suppkey = l_suppkey
        and l_orderkey = o_orderkey
        and o_custkey = c_custkey
        and c_nationkey = n_nationkey
        and n_regionkey = r_regionkey
        and o_orderdate between date '1995-01-01' and date '1996-12-31'
        and p_type = 'ECONOMY ANODIZED STEEL')
group by o_year
order by o_year;
```

Figura 22 – Consulta C5: Participação de mercado

5.2 Políticas de controle de acesso definidas

Um conjunto de políticas baseadas em regras de autorização foi definido com o intuito de explorar todas as possibilidades da proposta. A seguir são apresentadas as regras das políticas definidas:

R1. O gerente de vendas do norte da América e da Ásia deve ter acesso somente aos pedidos provenientes de fornecedores das nações do hemisfério Norte das regiões Ásia e América.

R2. Os clientes cadastrados devem ter acesso somente aos seus próprios pedidos.

R3. Os gestores de depósito de uma determinada nação só podem consultar fornecedores associados a sua própria nação.

R4. Os usuários do escritório de *marketing* dos Estados Unidos podem acessar os pedidos de clientes das nações Brasil e Argentina para realizar prospecção de produtos. Entretanto, vendedores só podem acessar os pedidos de clientes das suas próprias nações.

5.2.1 Implementação das políticas

A implementação das políticas definidas para os testes necessitou de algumas adaptações no modelo do TPC-H e no modelo do *framework*:

- Inclusão do campo *C_USR_CHAVE* na tabela *CUSTOMER* do TPC-H, referenciando o campo *USR_CHAVE* da tabela *USUARIO* do modelo do *framework*. Esse relacionamento denota o usuário (de banco) do cliente.
- Inclusão do campo *USR_NATIONKEY* na tabela *USUARIO* do modelo do *framework*, referenciando o campo *N_NATIONKEY* da tabela *NATION* do TPC-H. Esse relacionamento denota a nação do usuário.
- Inclusão do campo *N_HEMISPHERE* na tabela *NATION* do TPC-H, indicando o hemisfério onde a nação se localiza (*SOUTH*, *NORTH*, *BOTH*).

Em seguida, para cada política, foram construídos e cadastrados no modelo do *framework* os predicados para cada tabela afetada pela regra da política. Esses predicados foram então associados aos perfis de acordo com a listagem a seguir.

Observe que, para facilitar o entendimento de como o cadastro do predicado é feito, os mesmos receberam cores obedecendo a seguinte legenda: a parte em azul do predicado representa a regra cadastrada na tabela *REGRA*, a parte em verde representa o atributo do parâmetro cadastrado na tabela *ATRIBUTO*, a parte em laranja representa o operador cadastrado na tabela *OPERADOR*, a parte vermelha representa os valores cadastrados na tabela *VALOR_PARAMETRO* e, finalmente, a parte em preto representa o texto inserido pela função. Lembrando que a tabela *PARAMETRO* é responsável por criar o relacionamento entre a regra o atributo e o operador, portanto ela só é usada para fazer a junção dessas tabelas.

Política P1: baseada na regra R1, essa política dita que o gerente de vendas norte da América e da Ásia deve ter acesso somente aos pedidos provenientes de fornecedores das nações do hemisfério Norte das regiões Ásia e América.

Para implementar essa política foi definida uma hierarquia de perfis, com um perfil pai chamado *Gerente de Vendas* e um perfil filho chamado *Gerente de Vendas Norte América e Ásia*. As informações que devem ser protegidas por essa política são armazenadas nas tabelas *ORDERS* e *LINEITEM*, portanto, foi criado um predicado

para cada uma dessas tabelas, que são apresentados na Figura 23 e na Figura 24 respectivamente.

```
1=2 OR (O_CUSTKEY in (
select C_CUSTKEY
from TPCH.CUSTOMER
inner join TPCH.NATION on N_NATIONKEY = C_NATIONKEY
inner join TPCH.REGION on R_REGIONKEY = N_REGIONKEY
where R_NAME IN ('AMERICA', 'ASIA')
  and N_HEMISPHERE IN ('NORTH')))
```

Figura 23 - Predicado do perfil Gerente de Vendas Norte América e Ásia para a tabela *ORDERS*

```
1=2 OR (L_ORDERKEY in (
select O_ORDERKEY
from TPCH.ORDERS))
```

Figura 24 - Predicado do perfil Gerente de Vendas Norte América e Ásia para a tabela *LINEITEM*

Note que o predicado da tabela *ORDERS* possui dois parâmetros, onde para o primeiro foram definidos os valores *AMERICA* e *ASIA* e para o segundo foi definido o valor *NORTH*. Caso se deseje criar, por exemplo, um novo perfil filho na hierarquia chamado *Gerente de Vendas Sul África*, basta cadastrar para esse novo perfil o valor *AFRICA* no primeiro parâmetro e o valor *SOUTH* no segundo parâmetro.

Como a tabela *LINEITEM* referencia a tabela *ORDERS*, para aplicar a regra nessa primeira tabela basta criarmos um predicado que aponta para a segunda, pois assim o predicado da *ORDERS* será refletido diretamente na *LINEITEM*.

Política P2: baseada na regra R2, essa política dita que os clientes cadastros devem ter acesso somente aos seus próprios pedidos.

Para implementar essa política foi definido um perfil simples chamado *Cliente*. As informações que devem ser protegidas por essa política são armazenadas nas tabelas *ORDERS* e *LINEITEM*, portanto, foi criado um predicado para cada uma dessas tabelas, que são apresentados na Figura 25 e na Figura 26 respectivamente.

```
1=2 OR (O_CUSTKEY in (
select C_CUSTKEY
from TPCH.CUSTOMER
where C_USR_CHAVE = UPPER('?user')))
```

Figura 25 - Predicado do perfil Cliente para a tabela *ORDERS*

```
1=2 OR (L_ORDERKEY in (
select O_ORDERKEY
from TPCH.ORDERS))
```

Figura 26 - Predicado do perfil Cliente para a tabela *LINEITEM*

Note que, como o perfil *Cliente* é um perfil simples, o predicado da tabela *ORDERS* é inteiro cadastrado na tabela *REGRA*. Isso acontece porque o valor do único parâmetro desse predicado é o coringa *?user*, que será substituído em tempo de execução pelo nome do usuário de banco que realiza a operação. Dessa forma, não há necessidade de cadastrar valores para o parâmetro, portanto ele pode ser fixo.

O predicado para a tabela *LINEITEM* do perfil *Cliente* é o mesmo do perfil *Gerente de Vendas*. Lembrando que, como as tabelas *REGRA* e *PERFIL* possuem um relacionamento M × N, a mesma entrada da tabela *REGRA* foi aproveitada nos dois perfis.

Política P3: baseada na regra R3, essa política dita que os gestores de depósito de uma determinada nação só podem consultar fornecedores associados à sua própria nação.

Para implementar essa política foi definida uma hierarquia de perfis, com um perfil pai chamado *Gestor de Depósito* e um perfil filho chamado *Gestor de Depósito Local*. As informações que devem ser protegidas por essa política são armazenadas nas tabelas *SUPPLIER* e *PARTSUPP*, portanto, foi criado um predicado para cada uma dessas tabelas, que são apresentados na Figura 27 e na Figura 28 respectivamente.

```
1=2 OR (S_NATIONKEY in (
select USR_NATIONKEY
from FARBAC.USUARIO
where USR_CHAVE = UPPER('?user')))
```

Figura 27 - Predicado do perfil Gestor de Depósito Local para a tabela *SUPPLIER*

```
1=2 OR (PS_SUPPKEY in (
select S_SUPPKEY
from TPCH.SUPPLIER))
```

Figura 28 - Predicado do perfil Gestor de Depósito Local para a tabela *PARTSUPP*

Note que no predicado da tabela *SUPPLIER* novamente o coringa *?user* foi utilizado, mas dessa vez como valor de um parâmetro dinâmico (cadastrado na tabela *VALOR_PARAMETRO*), e o valor ainda foi definido como uma expressão, pois

utiliza a função *UPPER*. Esse predicado também poderia ter sido cadastrado inteiro da tabela *REGRA* como foi feito na política P2, contudo, preferimos cadastrá-lo dessa forma para exemplificar outra possibilidade de se cadastrar uma regra no *framework*.

Essa regra mostra como o predicado pode ser flexível o suficiente para utilizar informações de tabelas de domínios diferentes do domínio da tabela sendo protegida. Nesse caso a tabela protegida é do domínio TPCH e o predicado acessa uma tabela do domínio FARBAC.

Como a tabela *PARTSUPP* referencia a tabela *SUPPLIER*, para aplicar a regra na tabela *PARTSUPP* basta criarmos um predicado que aponta para a tabela *SUPPLIER*, refletindo assim o predicado definido na última.

Política P4: baseada na regra R4, essa política dita que os usuários do escritório de *marketing* dos Estados Unidos podem acessar os pedidos de clientes das nações Brasil e Argentina para realizar prospecção de produtos. Entretanto, vendedores só podem acessar os pedidos de clientes das suas próprias nações.

Para implementar essa política foi definida uma hierarquia de perfis, com um perfil pai chamado *Funcionário Marketing* e um perfil filho chamado *Funcionário Marketing para Brasil e Argentina*. Além disso, foi criado um perfil simples chamado *Funcionário Vendedor*. As informações que devem ser protegidas por essa política são armazenadas nas tabelas *ORDERS* e *LINEITEM*, portanto, foram criados predicados para cada perfil em cada uma dessas tabelas. Os predicados para o perfil *Funcionário Marketing para Brasil e Argentina* são apresentados na Figura 29 e na Figura 30, e os predicados para o perfil *Funcionário Vendedor* são apresentados na Figura 31 e na Figura 32.

```
1=2 OR (O_CUSTKEY in (
select C_CUSTKEY
from TPCH. CUSTOMER
inner join TPCH.NATION on N_NATIONKEY = C_NATIONKEY
where N_NAME IN ('BRAZIL', 'ARGENTINA')))
```

Figura 29 - Predicado do perfil Funcionário Marketing para Brasil e Argentina para a tabela *ORDERS*

```
1=2 OR (L_ORDERKEY in (
select O_ORDERKEY
```

```
from TPCH.ORDERS))
```

Figura 30 - Predicado do perfil Funcionário Marketing para Brasil e Argentina para a tabela *LINEITEM*

```
1=2 OR (O_CUSTKEY in (
select C_CUSTKEY
from TPCH.CUSTOMER
inner join FARBAC.USUARIO on USR_NATIONKEY = C_NATIONKEY
where USR_CHAVE = UPPER('?user')))
```

Figura 31 - Predicado do perfil Funcionário Vendedor para a tabela *ORDERS*

```
1=2 OR (L_ORDERKEY in (
select O_ORDERKEY
from TPCH.ORDERS))
```

Figura 32 - Predicado do perfil Funcionário Vendedor para a tabela *LINEITEM*

Note que novamente o predicado da tabela *LINEITEM* para ambos os perfis é o mesmo do perfil *Gerente de Vendas*, portanto mais uma vez a mesma entrada na tabela *REGRA* foi reutilizada.

5.3 Testes experimentais

O *framework* foi avaliado para seleções utilizando as consultas do TPC-H e para operações de *Insert*, *Update*, *Delete*, *Merge* e execução de *Triggers*. Os testes foram executados sobre o Oracle versão 10.2.0.3.0 através das aplicações Oracle SQL*Plus 10.2.0.3.0 e Oracle SQL Developer 2.1.0.63.

Um conjunto de usuários de banco foi criado e associado a cada perfil definido para as políticas, de acordo com a seguinte lista: *Bob* é o gerente de vendas norte da América e da Ásia; *John* é um cliente do Brasil; *Alison* é a gestora de depósito do Canadá; *Carol* é uma funcionária de marketing dos Estados Unidos; e *Paul* é um vendedor dos Estados Unidos.

Também foi criada a usuária *Alice*, que é a presidente da empresa e recebeu o privilégio *EXEMPT ACCESS POLICY*, que faz com que todas as políticas sejam ignoradas. Todos os testes foram feitos com a usuária *Alice* e com um dos outros usuários, dessa forma, facilmente visualizamos a realização do controle de acesso ao perceber que *Alice* não é submetida a esse controle, enquanto os outros usuários são.

5.3.1 Select

Os testes de seleção foram executados utilizando as consultas do TPC-H definidas na seção 5.1.2. A primeira consulta (C1) quantifica o aumento da receita resultante da eliminação de algum desconto percentual em determinado ano. Essa é uma consulta simples, que utiliza a função de agregação *SUM* e acessa somente a tabela *LINEITEM*. Para esse teste utilizamos os usuários *Alice* e *Bob*, e o resultado da execução da consulta por cada um deles pode ser visto na Figura 33 e na Figura 34 respectivamente.

```
SQL> connect alice
Informe a senha:*************
Conectado.
SQL> select sum(l_extendedprice * l_discount) as revenue
  2  from tpch.lineitem
  3  where l_shipdate >= date '1994-01-01'
  4    and l_shipdate < date '1994-01-01' + interval '1' year
  5    and l_discount between 0.06 - 0.01 and 0.06 + 0.01
  6    and l_quantity < 24;

   REVENUE
----------
88894386,6
```

Figura 33 - Resultado da execução da consulta C1 pela usuária *Alice*

```
SQL> connect bob
Informe a senha:*************
Conectado.
SQL> select sum(l_extendedprice * l_discount) as revenue
  2  from tpch.lineitem
  3  where l_shipdate >= date '1994-01-01'
  4    and l_shipdate < date '1994-01-01' + interval '1' year
  5    and l_discount between 0.06 - 0.01 and 0.06 + 0.01
  6    and l_quantity < 24;

   REVENUE
----------
21445915,5
```

Figura 34 - Resultado da execução da consulta C1 pelo usuário *Bob*

Como *Bob* é o gerente de vendas norte da América e da Ásia, ele tem acesso apenas aos itens de pedido do norte dessas regiões, enquanto *Alice*, que é a presidente, tem acesso a todos os itens de pedido. Para *Alice*, a mudança na receita é de 88.894.386,6, já para *Bob* é de apenas 21.445.915,5, afinal a sua visão é somente dos itens de pedido da América e da Ásia

A segunda consulta (C2) determina o fornecedor que deverá ser selecionado para fornecer uma determinada peça em uma determinada região. Essa é uma consulta com sub-consulta, que acessa as tabelas *PART*, *SUPPLIER*, *PARTSUPP*, *NATION* e *REGION*. Para esse teste utilizamos as usuárias *Alice* e *Alison*, e o resultado da execução da consulta por cada uma delas pode ser visto na Figura 35 e na Figura 36 respectivamente.

```
   -696,97 Supplier#000009155        PERU
   -797,25 Supplier#000005118        ARGENTINA
   -810,17 Supplier#000009641        UNITED STATES
   -846,13 Supplier#000001471        BRAZIL
   -846,13 Supplier#000001471        BRAZIL
    -859,7 Supplier#000003902        CANADA
   -889,35 Supplier#000003325        BRAZIL
   -926,61 Supplier#000001582        BRAZIL
   -926,61 Supplier#000001582        BRAZIL
    -951,7 Supplier#000000873        ARGENTINA
   -967,88 Supplier#000006916        UNITED STATES
   -990,16 Supplier#000005298        BRAZIL

447 linhas selecionadas.
```

Figura 35 - Resultado da execução da consulta C2 pela usuária *Alice*

```
   1249,81 Supplier#000003481        CANADA
   1130,03 Supplier#000001931        CANADA
   1130,03 Supplier#000001931        CANADA
    974,02 Supplier#000005516        CANADA
    958,07 Supplier#000000783        CANADA
    772,46 Supplier#000004014        CANADA
    764,57 Supplier#000006493        CANADA
    630,97 Supplier#000004304        CANADA
    580,84 Supplier#000002204        CANADA
     53,81 Supplier#000003625        CANADA
     46,35 Supplier#000001855        CANADA
    -859,7 Supplier#000003902        CANADA

100 linhas selecionadas.
```

Figura 36 - Resultado da execução da consulta C2 pela usuária *Alison*

Como *Alison* é a gestora de depósito do Canadá, ela tem acesso apenas aos fornecedores dessa nação, enquanto *Alice*, que é a presidente, tem acesso a todos os fornecedores. Para *Alice*, existem 447 fornecedores para a região America, já para *Alison* existem apenas 100, afinal a sua visão é somente dos fornecedores do Canadá.

A terceira consulta (C3) lista a quantidade de itens vendidos, remetidos e retornados. Essa é uma consulta com agrupamento, que acessa somente a tabela *LINEITEM*. Para esse teste utilizamos as usuárias *Alice* e *Carol*, e o resultado da

execução da consulta por cada uma delas pode ser visto na Figura 37 e na Figura 38 respectivamente.

```
SQL> connect alice
Informe a senha:************
Conectado.
SQL> select
  2    l_returnflag,
  3    l_linestatus,
  4    sum(l_quantity) as sum_qty,
  5    sum(l_extendedprice) as sum_base_price,
  6    sum(l_extendedprice*(1-l_discount)) as sum_disc_price,
  7    sum(l_extendedprice*(1-l_discount)*(1+l_tax)) as sum_charge,
  8    avg(l_quantity) as avg_qty,
  9    avg(l_extendedprice) as avg_price,
 10    avg(l_discount) as avg_disc,
 11    count(*) as count_order
 12  from tpch.lineitem
 13  where l_shipdate <= date '1998-12-01' - interval '90' day (3)
 14  group by l_returnflag,l_linestatus
 15  order by l_returnflag,l_linestatus;

L L    SUM_QTY SUM_BASE_PRICE SUM_DISC_PRICE SUM_CHARGE    AVG_QTY
- - ---------- -------------- -------------- ---------- ----------
A F   27221567     4,0819E+10     3,8780E+10 4,0331E+10 25,5111945
N F     713813     1070192107     1016695666 1057404887 25,4987855
N O   53768078     8,0644E+10     7,6612E+10 7,9680E+10 25,4989851
R F   27217676     4,0820E+10     3,8781E+10 4,0331E+10 25,5066874
```

Figura 37 - Resultado da execução da consulta C3 pela usuária *Alice*

```
SQL> connect carol
Informe a senha:*************
Conectado.
SQL> select
  2    l_returnflag,
  3    l_linestatus,
  4    sum(l_quantity) as sum_qty,
  5    sum(l_extendedprice) as sum_base_price,
  6    sum(l_extendedprice*(1-l_discount)) as sum_disc_price,
  7    sum(l_extendedprice*(1-l_discount)*(1+l_tax)) as sum_charge,
  8    avg(l_quantity) as avg_qty,
  9    avg(l_extendedprice) as avg_price,
 10    avg(l_discount) as avg_disc,
 11    count(*) as count_order
 12  from tpch.lineitem
 13  where l_shipdate <= date '1998-12-01' - interval '90' day (3)
 14  group by l_returnflag,l_linestatus
 15  order by l_returnflag,l_linestatus;

L L    SUM_QTY SUM_BASE_PRICE SUM_DISC_PRICE SUM_CHARGE    AVG_QTY
- - ---------- -------------- -------------- ---------- ----------
A F    2165667     3250349173     3088025318 3211563375 25,4856312
N F      56919     85273288,5     81038833,4 84306917,5 25,1965471
N O    4301729     6450418974     6127783315 6372515456 25,5097165
R F    2190232     3281731929     3117807332 3242240787 25,5653189
```

Figura 38 - Resultado da execução da consulta C3 pela usuária *Carol*

Como *Carol* é uma funcionária de *marketing* dos Estados Unidos, ela pode acessar os itens de pedidos de clientes das nações Brasil e Argentina para realizar prospecção de produtos, enquanto *Alice*, que é a presidente, tem acesso a todos os itens de pedidos. Pela contagem de itens de pedidos (*SUM_QTY*) pode-se perceber que *Alice* visualiza muito mais itens do que *Carol*, já que *Carol* tem visão somente dos itens de pedido do Brasil e da Argentina.

A quarta consulta (C4) ranqueia os consumidores que realizaram pedidos de grande quantidade, de acordo com o preço total de cada pedido. Essa é uma consulta com sub-consulta, com agrupamento e com filtro de agrupamento, que acessa as tabelas *CUSTOMER*, *ORDERS* e *LINEITEM*. Para esse teste utilizamos os usuários *Alice*, *Carol* e *Paul*, e o resultado da execução da consulta por cada uma deles pode ser visto na Figura 39, na Figura 40 e na Figura 41 respectivamente.

```
Customer#000141823          141823    2806245 29/12/96       446269             310
Customer#000053029           53029    2662214 13/08/93       446144             302
Customer#000018188           18188    3037414 25/01/95       443807             308
Customer#000066533           66533      29158 21/10/95       443577             305
Customer#000037729           37729    4134341 29/06/95       441083             309
Customer#000003566            3566    2329187 04/01/98       439803             304
Customer#000119989          119989    1544643 20/09/97       434568             320
Customer#000003680            3680    3861123 03/07/98       433526             301
Customer#000113131          113131     967334 15/12/95       432958             301
Customer#000141098          141098     565574 24/09/95       430987             301
Customer#000015631           15631    1845057 12/05/94       419880             302
Customer#000012599           12599    4259524 12/02/98       415201             304
Customer#000069904           69904    1742403 19/10/96       408513             305
Customer#000017746           17746       6882 09/04/97       408447             303
Customer#000013072           13072    1481925 15/03/98       399195             301
Customer#000082441           82441     857959 07/02/94       382580             305
Customer#000088703           88703    2995076 30/01/94       363812             302

39 linhas selecionadas.
```

Figura 39 - Resultado da execução da consulta C4 pela usuária *Alice*

```
SQL> connect carol
Informe a senha:************
Conectado.
SQL> select c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice, sum(l_quantity)
  2  from tpch.customer, tpch.orders, tpch.lineitem
  3  where o_orderkey in (select l_orderkey
  4                       from tpch.lineitem
  5                       group by l_orderkey having sum(l_quantity) > 300)
  6    and c_custkey = o_custkey
  7    and o_orderkey = l_orderkey
  8  group by c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice
  9  order by o_totalprice desc, o_orderdate;

C_NAME                    C_CUSTKEY O_ORDERKEY O_ORDERD O_TOTALPRICE SUM(L_QUANTITY)
------------------------- ---------- ---------- -------- ------------ ---------------
Customer#000144617            144617    3043270 12/02/97       530604             317
Customer#000003566              3566    2329187 04/01/98       439803             304
```

Figura 40 - Resultado da execução da consulta C4 pela usuária *Carol*

```
SQL> connect paul
Informe a senha:************
Conectado.
SQL> select c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice, sum(l_quantity)
  2  from tpch.customer, tpch.orders, tpch.lineitem
  3  where o_orderkey in (select l_orderkey
  4                       from tpch.lineitem
  5                       group by l_orderkey having sum(l_quantity) > 300)
  6    and c_custkey = o_custkey
  7    and o_orderkey = l_orderkey
  8  group by c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice
  9  order by o_totalprice desc, o_orderdate;

C_NAME                    C_CUSTKEY O_ORDERKEY O_ORDERD O_TOTALPRICE SUM(L_QUANTITY)
------------------------- ---------- ---------- -------- ------------ ---------------
Customer#000105995            105995    2096705 03/07/94       469693             307
Customer#000088876             88876     983201 30/12/93       446717             304
```

Figura 41 - Resultado da execução da consulta C4 pelo usuário *Paul*

Como *Carol* é uma funcionária de *marketing* dos Estados Unidos, ela pode acessar os pedidos de clientes das nações Brasil e Argentina para realizar prospecção de produtos, já *Paul*, que é um vendedor dos Estados Unidos, pode acessar os pedidos de clientes dos Estados Unidos, e *Alice*, que é a presidente, tem acesso a todos os pedidos. Para *Alice*, existem 39 consumidores que realizaram pedidos de

grande quantidade, já para *Carol* e *Paul* existem apenas 2 que, além disso, são diferentes para cada um, afinal a visão de *Carol* é sobre os pedidos do Brasil e da Argentina e a do *Paul* é sobre os pedidos dos Estados unidos.

Imagine agora que *Paul* seja realocado para o setor de *Marketing*, mas continue realizando vendas. Nesse novo cenário *Paul* possui dois perfis: *Funcionário Marketing para Brasil e Argentina* e *Funcionário Vendedor*. A consulta C4 foi então repetida pelo usuário e o resultado é apresentado na Figura 42.

```
SQL> connect paul
Informe a senha:*************
Conectado.
SQL> select c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice, sum(l_quantity)
  2  from tpch.customer, tpch.orders, tpch.lineitem
  3  where o_orderkey in (select l_orderkey
  4                       from tpch.lineitem
  5                       group by l_orderkey having sum(l_quantity) > 300)
  6    and c_custkey = o_custkey
  7    and o_orderkey = l_orderkey
  8  group by c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice
  9  order by o_totalprice desc, o_orderdate;

C_NAME                    C_CUSTKEY O_ORDERKEY O_ORDERD O_TOTALPRICE SUM(L_QUANTITY)
------------------------- ---------- ---------- -------- ------------ ---------------
Customer#000144617            144617    3043270 12/02/97       530604             317
Customer#000105995            105995    2096705 03/07/94       469693             307
Customer#000088876             88876     983201 30/12/93       446717             304
Customer#000003566              3566    2329187 04/01/98       439803             304
```

Figura 42 - Resultado da execução da consulta C4 pelo usuário *Paul* após a realocação

Note que, agora que *Paul* possui os dois perfis, ele pode visualizar os pedidos do Brasil, da Argentina e dos Estados Unidos, pois a forma de combinação de perfis implementada neste trabalho foi permissiva, como ilustrado na seção 3.1 (Figura 5). Portanto, o novo resultado da consulta é a união do resultado anterior com o resultado da *Carol*, ou seja, a união dos perfis *Funcionário Marketing para Brasil e Argentina* e *Funcionário Vendedor*.

A quinta consulta (C5) determina a mudança na participação de mercado de uma determinada região para um tipo de peça durante dois anos. Essa é uma consulta com sub-consulta, com agrupamento e com a função de agregação *SUM* em conjunto com a operação *CASE*, que acessa as tabelas *PART*, *SUPPLIER*, *LINEITEM*, *ORDERS*, *CUSTOMER*, *NATION* e *REGION*. Para esse teste utilizamos os usuários *Alice* e *Bob*, e o resultado da execução da consulta por cada um deles pode ser visto na Figura 43 e na Figura 44 respectivamente.

Note que, nesse momento, existe uma política aplicada sobre a tabela *SUPPLIER*, e Bob não possui nenhum perfil permitindo acesso sobre essa tabela,

portanto ele não pode visualizar nenhum fornecedor. Por praticidade essa política foi desativada durante esse teste, mas em um cenário real o correto seria criar um perfil para *Bob* liberando o acesso aos fornecedores.

```
SQL> connect alice
Informe a senha:************
Conectado.
SQL> select o_year,
  2     sum(case
  3         when region = 'AMERICA'
  4         then volume
  5         else 0
  6         end) / sum(volume) as mkt_share
  7  from (select extract(year from o_orderdate) as o_year,
  8          l_extendedprice * (1-l_discount) as volume,
  9          r_name as region
 10        from tpch.part, tpch.supplier, tpch.lineitem, tpch.orders,
 11             tpch.customer, tpch.nation, tpch.region
 12        where p_partkey = l_partkey
 13          and s_suppkey = l_suppkey
 14          and l_orderkey = o_orderkey
 15          and o_custkey = c_custkey
 16          and c_nationkey = n_nationkey
 17          and n_regionkey = r_regionkey
 18          and o_orderdate between date '1995-01-01' and date '1996-12-31'
 19          and p_type = 'ECONOMY ANODIZED STEEL')
 20  group by o_year
 21  order by o_year;

    O_YEAR  MKT_SHARE
---------- ----------
      1995  ,19573373
      1996 ,184147628
```

Figura 43 - Resultado da execução da consulta C5 pela usuária *Alice*

```
SQL> connect bob
Informe a senha:************
Conectado.
SQL> select o_year,
  2     sum(case
  3         when region = 'AMERICA'
  4         then volume
  5         else 0
  6         end) / sum(volume) as mkt_share
  7  from (select extract(year from o_orderdate) as o_year,
  8          l_extendedprice * (1-l_discount) as volume,
  9          r_name as region
 10        from tpch.part, tpch.supplier, tpch.lineitem, tpch.orders,
 11             tpch.customer, tpch.nation, tpch.region
 12        where p_partkey = l_partkey
 13          and s_suppkey = l_suppkey
 14          and l_orderkey = o_orderkey
 15          and o_custkey = c_custkey
 16          and c_nationkey = n_nationkey
 17          and n_regionkey = r_regionkey
 18          and o_orderdate between date '1995-01-01' and date '1996-12-31'
 19          and p_type = 'ECONOMY ANODIZED STEEL')
 20  group by o_year
 21  order by o_year;

    O_YEAR  MKT_SHARE
---------- ----------
      1995 ,318472238
      1996 ,321055692
```

Figura 44 - Resultado da execução da consulta C5 pelo usuário *Bob*

Como *Bob* é o gerente de vendas norte da América e da Ásia, ele tem acesso apenas aos itens de pedido do norte dessas regiões, enquanto *Alice*, que é a

presidente, tem acesso a todos os itens de pedido. Para *Alice*, a participação da região América no mercado para esse tipo de peça foi de aproximadamente 19,5% em 1995 e de aproximadamente 18,4% em 1996. Já para *Bob*, essas participações foram de aproximadamente 31,8% em 1995 e de aproximadamente 32,1% em 1996.

Note que como *Alice* tem uma visão global, a consulta retorna o percentual de participação em relação a todas as outras regiões, enquanto que para *Bob* a consulta retorna o percentual de participação em relação somente às regiões que ele é responsável, ou seja, norte da América e da Ásia. Esse é exatamente o tipo de consulta que pode gerar mal entendidos em uma abordagem que segue o *Truman Model*. Se *Bob* não souber que esse percentual é em relação somente às regiões norte da América e da Ásia, ele pode entender que a região sob sua responsabilidade está com um índice de participação de mercado alto, enquanto na verdade está abaixo da média (a média é 20% pois existem cinco regiões). Para evitar esse tipo de problema todos os usuários devem entender muito bem a sua visão dos dados.

5.3.2 Insert

Para testar a inserção utilizamos os usuários *John*, que é um cliente do Brasil, e *Alice*, que é a presidente da empresa. Os comandos desse teste realizam inserções na tabela *ORDERS*, primeiramente com o identificador correto para o usuário *John* (*O_CUSTKEY = 92*), e em seguida com o identificador de outro usuário (*O_CUSTKEY = 100*), também cliente da empresa. Os resultados da execução de ambos os comandos são apresentados na Figura 45 e na Figura 46 respectivamente.

```
SQL> connect john
Informe a senha:************
Conectado.
SQL> insert into tpch.orders (o_orderkey, o_custkey, o_orderstatus,
  2                           o_totalprice, o_orderdate, o_orderpriority,
  3                           o_clerk, o_shippriority, o_comment)
  4                    values (6000001, 92, 'F', 150000,
  5                           sysdate, '4-NOT SPECIFIED', 'Clerk#000000979',
  6                           0, 'teste de insert');

1 linha criada.

SQL> select * from tpch.orders;

O_ORDERKEY  O_CUSTKEY O O_TOTALPRICE O_ORDERD O_ORDERPRIORITY O_CLERK
---------- ---------- - ------------ -------- --------------- ---------------
    223687         92 F       208229 08/07/93 3-MEDIUM        Clerk#000000639
    760615         92 F       301485 29/08/94 1-URGENT        Clerk#000000773
   1519395         92 O       218181 03/01/98 5-LOW           Clerk#000000360
   1795040         92 F        13393 03/03/92 2-HIGH          Clerk#000000070
   2351813         92 F        92296 14/02/95 5-LOW           Clerk#000000647
   2522405         92 O       298230 17/08/97 1-URGENT        Clerk#000000649
   2561285         92 F        73741 08/12/94 5-LOW           Clerk#000000979
   2575524         92 O       219189 04/05/96 2-HIGH          Clerk#000000423
   3057349         92 F       181145 29/03/95 4-NOT SPECIFIED Clerk#000000379
   3647366         92 O       258650 01/09/95 4-NOT SPECIFIED Clerk#000000721
   3951590         92 O       118910 06/06/96 1-URGENT        Clerk#000000978
   4593444         92 F       257031 16/08/92 4-NOT SPECIFIED Clerk#000000423
   4810978         92 F       284793 28/02/92 4-NOT SPECIFIED Clerk#000000743
   5293862         92 O       197231 02/02/98 2-HIGH          Clerk#000000092
   5653925         92 O       186931 18/06/97 3-MEDIUM        Clerk#000000452
   5971713         92 F       149395 30/05/93 4-NOT SPECIFIED Clerk#000000051
   6000001         92 F       150000 18/10/10 4-NOT SPECIFIED Clerk#000000979

17 linhas selecionadas.
```

Figura 45 - Resultado da inserção realizada pelo usuário *John* com o identificador correto

```
SQL> connect john
Informe a senha:************
Conectado.
SQL> insert into tpch.orders (o_orderkey, o_custkey, o_orderstatus,
  2                           o_totalprice, o_orderdate, o_orderpriority,
  3                           o_clerk, o_shippriority, o_comment)
  4                   values (6000002, 100, 'F', 150000,
  5                           sysdate, '4-NOT SPECIFIED', 'Clerk#000000979',
  6                           0, 'teste de insert');
insert into tpch.orders (o_orderkey, o_custkey, o_orderstatus,
                *
ERRO na linha 1:
ORA-28115: política com violação de opção de verificação


SQL> select * from tpch.orders;

O_ORDERKEY  O_CUSTKEY O O_TOTALPRICE O_ORDERD O_ORDERPRIORITY O_CLERK
---------- ---------- - ------------ -------- --------------- ---------------
    223687         92 F       208229 08/07/93 3-MEDIUM        Clerk#000000639
    760615         92 F       301485 29/08/94 1-URGENT        Clerk#000000773
   1519395         92 O       218181 03/01/98 5-LOW           Clerk#000000360
   1795040         92 F        13393 03/03/92 2-HIGH          Clerk#000000070
   2351813         92 F        92296 14/02/95 5-LOW           Clerk#000000647
   2522405         92 O       298230 17/08/97 1-URGENT        Clerk#000000649
   2561285         92 F        73741 08/12/94 5-LOW           Clerk#000000979
   2575524         92 O       219189 04/05/96 2-HIGH          Clerk#000000423
   3057349         92 F       181145 29/03/95 4-NOT SPECIFIED Clerk#000000379
   3647366         92 O       258650 01/09/95 4-NOT SPECIFIED Clerk#000000721
   3951590         92 O       118910 06/06/96 1-URGENT        Clerk#000000978
   4593444         92 F       257031 16/08/92 4-NOT SPECIFIED Clerk#000000423
   4810978         92 F       284793 28/02/92 4-NOT SPECIFIED Clerk#000000743
   5293862         92 O       197231 02/02/98 2-HIGH          Clerk#000000092
   5653925         92 O       186931 18/06/97 3-MEDIUM        Clerk#000000452
   5971713         92 F       149395 30/05/93 4-NOT SPECIFIED Clerk#000000051
   6000001         92 F       150000 18/10/10 4-NOT SPECIFIED Clerk#000000979

17 linhas selecionadas.
```

Figura 46 - Resultado da tentativa de inserção realizada pelo usuário *John* com o identificador incorreto

O resultado mostra que o usuário foi capaz de inserir um pedido em seu nome (pedido número 6000001), enquanto foi impedido de inserir um pedido em nome de outro cliente (pedido número 6000002). Contudo, é preciso salientar que esse teste foi realizado com uma política que foi aplicada com a opção de *UPDATE_CHECK* ativada, portanto, para realizar uma avaliação mais completa, a política foi reaplicada com a opção de *UPDATE_CHECK* desativada e o segundo comando foi repetido. O resultado da re-execução é apresentado na Figura 47.

```
SQL> connect john
Informe a senha:************
Conectado.
SQL> insert into tpch.orders (o_orderkey, o_custkey, o_orderstatus,
  2                           o_totalprice, o_orderdate, o_orderpriority,
  3                           o_clerk, o_shippriority, o_comment)
  4                   values (6000002, 100, 'F', 150000,
  5                           sysdate, '4-NOT SPECIFIED', 'Clerk#000000979',
  6                           0, 'teste de insert');

1 linha criada.

SQL> select * from tpch.orders where o_orderkey = 6000002;

não há linhas selecionadas

SQL> connect alice
Informe a senha:************
Conectado.
SQL> select * from tpch.orders where o_orderkey = 6000002;

O_ORDERKEY  O_CUSTKEY O O_TOTALPRICE O_ORDERD O_ORDERPRIORITY O_CLERK
---------- ---------- - ------------ -------- --------------- ---------------
   6000002        100 F       150000 18/10/10 4-NOT SPECIFIED Clerk#000000979
```

Figura 47 - Resultado da inserção realizada pelo usuário *John* com o identificador incorreto e com a opção de *UPDATE_CHECK* desativada

Note que, nessa segunda execução, o usuário *John* foi capaz de inserir um registro que ele mesmo não tem acesso depois de inserido. Isso acontece, pois é a opção de *UPDATE_CHECK* que verifica se um usuário terá acesso aos dados que ele tenta inserir, portanto, se ela não estiver ativada, a regras não são aplicadas em inserções.

5.3.3 Update

Para testar a atualização, novamente utilizamos os usuários *John* e *Alice*. Os comandos desse teste realizam atualizações na tabela *ORDERS*, atualizando um pedido realizado pelo usuário *John* (*O_ORDERKEY = 5971713*), e em seguida atualizando um pedido realizado por outro usuário (*O_ORDERKEY = 5877348*). Os resultados da execução de ambos os comandos são apresentados na Figura 48 e na Figura 49 respectivamente.

```
SQL> connect john
Informe a senha:*************
Conectado.
SQL> select o_orderkey, o_custkey, o_orderpriority
  2  from tpch.orders
  3  where o_orderkey = 5971713;

O_ORDERKEY  O_CUSTKEY O_ORDERPRIORITY
---------- ---------- ---------------
   5971713         92 4-NOT SPECIFIED

SQL> update tpch.orders
  2  set o_orderpriority = '3-MEDIUM'
  3  where o_orderkey = 5971713;

1 linha atualizada.

SQL> select o_orderkey, o_custkey, o_orderpriority
  2  from tpch.orders
  3  where o_orderkey = 5971713;

O_ORDERKEY  O_CUSTKEY O_ORDERPRIORITY
---------- ---------- ---------------
   5971713         92 3-MEDIUM
```

Figura 48 - Resultado da atualização realizada pelo usuário *John* para um pedido realizado por si mesmo

```
SQL> connect john
Informe a senha:*************
Conectado.
SQL> select o_orderkey, o_custkey, o_orderpriority
  2  from tpch.orders
  3  where o_orderkey = 5877348;

não há linhas selecionadas

SQL> update tpch.orders
  2  set o_orderpriority = '3-MEDIUM'
  3  where o_orderkey = 5877348;

0 linhas atualizadas.

SQL> select o_orderkey, o_custkey, o_orderpriority
  2  from tpch.orders
  3  where o_orderkey = 5877348;

não há linhas selecionadas

SQL> connect alice
Informe a senha:*************
Conectado.
SQL> select o_orderkey, o_custkey, o_orderpriority
  2  from tpch.orders
  3  where o_orderkey = 5877348;

O_ORDERKEY  O_CUSTKEY O_ORDERPRIORITY
---------- ---------- ---------------
   5877348        100 1-URGENT

SQL> update tpch.orders
  2  set o_orderpriority = '3-MEDIUM'
  3  where o_orderkey = 5877348;

1 linha atualizada.

SQL> select o_orderkey, o_custkey, o_orderpriority
  2  from tpch.orders
  3  where o_orderkey = 5877348;

O_ORDERKEY  O_CUSTKEY O_ORDERPRIORITY
---------- ---------- ---------------
   5877348        100 3-MEDIUM
```

Figura 49 - Resultado da tentativa de atualização realizada pelo usuário *John* para um pedido realizado por outro usuário e, em seguida, resultado com atualização realizada por Alice

O resultado mostra que o usuário foi capaz de atualizar um pedido em seu nome, enquanto foi impedido de atualizar um pedido em nome de outro cliente. Imagine agora que *John* tenta transferir um pedido em seu nome para outro cliente, alterando o identificador do cliente (*O_CUSTKEY*). Em outras palavras, imagine que

John tenta alterar um registro que ele atualmente tem acesso para um estado que ele não terá acesso. A Figura 50 mostra o resultado dessa tentativa.

```
SQL> connect john
Informe a senha:************
Conectado.
SQL> select o_orderkey, o_custkey, o_orderpriority
  2  from tpch.orders
  3  where o_orderkey = 5971713;

O_ORDERKEY  O_CUSTKEY O_ORDERPRIORITY
---------- ---------- ---------------
   5971713         92 4-NOT SPECIFIED

SQL> update tpch.orders
  2  set o_custkey = 100
  3  where o_orderkey = 5971713;
update tpch.orders
            *
ERRO na linha 1:
ORA-28115: política com violação de opção de verificação


SQL> select o_orderkey, o_custkey, o_orderpriority
  2  from tpch.orders
  3  where o_orderkey = 5971713;

O_ORDERKEY  O_CUSTKEY O_ORDERPRIORITY
---------- ---------- ---------------
   5971713         92 4-NOT SPECIFIED
```

Figura 50 - Resultado da tentativa de transferência de pedido realizada pelo usuário *John*

Mais uma vez o resultado mostrou que o usuário não foi capaz de atualizar um pedido em seu nome para um estado que ele não tenha acesso. Contudo, novamente é preciso salientar que esse teste foi realizado com uma política que foi aplicada com a opção de *UPDATE_CHECK* ativada. Portanto, para realizar uma avaliação mais completa, a política foi reaplicada com a opção de *UPDATE_CHECK* desativada e a tentativa de atualização foi repetida. O resultado da nova tentativa é apresentado na Figura 51.

```
SQL> connect john
Informe a senha:*************
Conectado.
SQL> select o_orderkey, o_custkey, o_orderpriority
  2  from tpch.orders
  3  where o_orderkey = 5971713;

O_ORDERKEY  O_CUSTKEY O_ORDERPRIORITY
---------- ---------- ---------------
   5971713         92 4-NOT SPECIFIED

SQL> update tpch.orders
  2  set o_custkey = 100
  3  where o_orderkey = 5971713;

1 linha atualizada.

SQL> select o_orderkey, o_custkey, o_orderpriority
  2  from tpch.orders
  3  where o_orderkey = 5971713;

não há linhas selecionadas

SQL> connect alice
Informe a senha:*************
Conectado.
SQL> select o_orderkey, o_custkey, o_orderpriority
  2  from tpch.orders
  3  where o_orderkey = 5971713;

O_ORDERKEY  O_CUSTKEY O_ORDERPRIORITY
---------- ---------- ---------------
   5971713        100 4-NOT SPECIFIED
```

Figura 51 - Resultado da tentativa de transferência de pedido realizada pelo usuário *John* com a opção de *UPDATE_CHECK* desativada

Nessa segunda tentativa, o usuário *John* foi capaz de atualizar um registro que ele mesmo não tem acesso depois de atualizado. Isso acontece, pois é a opção de *UPDATE_CHECK* que verifica se um usuário terá acesso aos dados que ele tenta atualizar depois da atualização. Portanto, se ela não estiver ativada, as regras não são aplicadas em resultados de atualizações. Observe que após a atualização do pedido, John não tem mais acesso a ele (vide quarto comando "SQL>").

Note que a opção de *UPDATE_CHECK* só afeta o resultado da atualização, ou seja, com a opção ativada ou não o usuário nunca pode atualizar um registro que ele atualmente não possui acesso.

5.3.4 Delete

Para testar a remoção, mais uma vez utilizamos os usuários *John* e *Alice*. Os comandos desse teste realizam remoções na tabela *ORDERS*, removendo um pedido realizado pelo usuário *John* (*O_ORDERKEY = 5971713*), e em seguida removendo um pedido realizado por outro usuário (*O_ORDERKEY = 5877348*). Os resultados da execução de ambos os comandos são apresentados na Figura 48 e na Figura 49, respectivamente.

```
SQL> connect john
Informe a senha:************
Conectado.
SQL> select o_orderkey, o_custkey, o_orderpriority
  2  from tpch.orders
  3  where o_orderkey = 5971713;

O_ORDERKEY  O_CUSTKEY O_ORDERPRIORITY
---------- ---------- ---------------
   5971713         92 4-NOT SPECIFIED

SQL> delete from tpch.orders where o_orderkey = 5971713;

1 linha deletada.

SQL> select o_orderkey, o_custkey, o_orderpriority
  2  from tpch.orders
  3  where o_orderkey = 5971713;

não há linhas selecionadas
```

Figura 52 - Resultado da remoção realizada pelo usuário *John* para um pedido realizado por si mesmo

```
SQL> connect john
Informe a senha:*************
Conectado.
SQL> select o_orderkey, o_custkey, o_orderpriority
  2  from tpch.orders
  3  where o_orderkey = 5877348;

não há linhas selecionadas

SQL> delete from tpch.orders where o_orderkey = 5877348;

0 linhas deletadas.

SQL> select o_orderkey, o_custkey, o_orderpriority
  2  from tpch.orders
  3  where o_orderkey = 5877348;

não há linhas selecionadas

SQL> connect alice
Informe a senha:*************
Conectado.
SQL> select o_orderkey, o_custkey, o_orderpriority
  2  from tpch.orders
  3  where o_orderkey = 5877348;

O_ORDERKEY  O_CUSTKEY O_ORDERPRIORITY
---------- ---------- ---------------
   5877348        100 1-URGENT

SQL> delete from tpch.orders where o_orderkey = 5877348;

1 linha deletada.

SQL> select o_orderkey, o_custkey, o_orderpriority
  2  from tpch.orders
  3  where o_orderkey = 5877348;

não há linhas selecionadas
```

Figura 53 - Resultado da tentativa de remoção realizada pelo usuário *John* para um pedido realizado por outro usuário, e em seguida, resultado da remoção realizada por Alice

O resultado mostra que o usuário foi capaz de remover um pedido em seu nome, enquanto foi impedido de remover um pedido em nome de outro cliente.

5.3.5 Merge

O *Merge* é uma operação que permite definir ao mesmo tempo atualizações, remoções e inserções em uma tabela, a partir das informações de outra tabela ou visão (entrada), como uma espécie de união entre a tabela original e as informações

de entrada. O *Merge* recebe o conjunto de informações de entrada e uma cláusula de verificação de existência para os registros da tabela que irá receber as informações. A partir dessa cláusula, se o registro existir, é feita uma atualização ou remoção e, se o registro não existir, é feita uma inserção. O Merge é uma operação padrão da SQL a partir da revisão de 2003 [Eisenberg *et al*., 2004]. Um exemplo de uso de merge é apresentado no terceiro comando da Figura 54.

Para testar o *Merge* utilizamos os usuários *Alison*, que é a gestora de depósito do Canadá, e *Alice*, que é a presidente da empresa. O comando utilizado faz o *Merge* na tabela *SUPPLIER* e recebe como informações de entrada os seguintes códigos de fornecedor: 9972 e 9974 que são códigos de fornecedores do Canadá, 9975 e 9976 que são códigos de fornecedores de outras nações e 10001 que não é código de nenhum fornecedor. A partir dessas informações o comando tenta atualizar o campo *S_COMMENT* dos registros que existem, tenta remover os registros 9972 e 9975 e tenta inserir um novo registro para o código do fornecedor que ainda não existe. O resultado da execução do comando por cada uma das usuárias é apresentado na Figura 54 e na Figura 55 respectivamente.

```
SQL> connect alison
Informe a senha:************
Conectado.
SQL> select s_suppkey, s_comment from tpch.supplier
  2  where s_suppkey = 9972
  3     or s_suppkey = 9974
  4     or s_suppkey = 9975
  5     or s_suppkey = 9976
  6     or s_suppkey = 10001;

 S_SUPPKEY S_COMMENT
---------- -------------------------------------------------------------------------------
      9972 slyly regular pains affix blithely. blithely express
      9974 boldly ironic dinos cajole along the i

SQL> merge into tpch.supplier s
  2  using (select 9972 as s_suppkey, 'Teste merge' as s_comment from dual union
  3         select 9974 as s_suppkey, 'Teste merge' as s_comment from dual union
  4         select 9975 as s_suppkey, 'Teste merge' as s_comment from dual union
  5         select 9976 as s_suppkey, 'Teste merge' as s_comment from dual union
  6         select 10001 as s_suppkey, 'Teste merge' as s_comment from dual) e
  7  on (s.s_suppkey = e.s_suppkey)
  8  when matched then
  9    update set s.s_comment = e.s_comment
 10    delete where s_suppkey = 9972
 11             or s_suppkey = 9975
 12  when not matched then
 13    insert values (e.s_suppkey, 'Supplier', 'Address', 3, '1234', 8000, e.s_comment);
merge into tpch.supplier s
           *
ERRO na linha 1:
ORA-28132: A intercalação na sintaxe não suporta políticas de segurança.


SQL> select s_suppkey, s_comment from tpch.supplier
  2  where s_suppkey = 9972
  3     or s_suppkey = 9974
  4     or s_suppkey = 9975
  5     or s_suppkey = 9976
  6     or s_suppkey = 10001;

 S_SUPPKEY S_COMMENT
---------- -------------------------------------------------------------------------------
      9972 slyly regular pains affix blithely. blithely express
      9974 boldly ironic dinos cajole along the i
```

Figura 54 - Resultado da tentativa de *Merge* realizado pela usuária *Alison*

```
SQL> connect alice
Informe a senha:************
Conectado.
SQL> select s_suppkey, s_comment from tpch.supplier
  2  where s_suppkey = 9972
  3     or s_suppkey = 9974
  4     or s_suppkey = 9975
  5     or s_suppkey = 9976
  6     or s_suppkey = 10001;

 S_SUPPKEY S_COMMENT
---------- -------------------------------------------------------------------------------
      9972 slyly regular pains affix blithely. blithely express
      9974 boldly ironic dinos cajole along the i
      9975 furiously pending packages up the
      9976 furiously final theodolites boost quickly regular, even packages.

SQL> merge into tpch.supplier s
  2  using (select 9972 as s_suppkey, 'Teste merge' as s_comment from dual union
  3         select 9974 as s_suppkey, 'Teste merge' as s_comment from dual union
  4         select 9975 as s_suppkey, 'Teste merge' as s_comment from dual union
  5         select 9976 as s_suppkey, 'Teste merge' as s_comment from dual union
  6         select 10001 as s_suppkey, 'Teste merge' as s_comment from dual) e
  7  on (s.s_suppkey = e.s_suppkey)
  8  when matched then
  9    update set s.s_comment = e.s_comment
 10    delete where s_suppkey = 9972
 11              or s_suppkey = 9975
 12  when not matched then
 13    insert values (e.s_suppkey, 'Supplier', 'Address', 3, '1234', 8000, e.s_comment);

5 linhas intercaladas.

SQL> select s_suppkey, s_comment from tpch.supplier
  2  where s_suppkey = 9972
  3     or s_suppkey = 9974
  4     or s_suppkey = 9975
  5     or s_suppkey = 9976
  6     or s_suppkey = 10001;

 S_SUPPKEY S_COMMENT
---------- -------------------------------------------------------------------------------
      9974 Teste merge
      9976 Teste merge
     10001 Teste merge
```

Figura 55 - Resultado do *Merge* realizado pela usuária *Alice*

O comando realizado pela usuária *Alison* não foi executado corretamente, pois o Oracle ainda não suporta a realização de *Merge* em tabelas que possuem controle de acesso. Como não é possível utilizar o merge em conjunto com o VPD, o próprio Oracle recomenda que sejam usados comandos *Insert* e *Update* equivalentes para garantir o controle de acesso [Jeloka, 2008]. Contudo, note que quando realizado pela usuária Alice, que possui o privilégio *EXEMPT ACCESS POLICY*, o comando é executado normalmente.

5.3.6 Trigger

A *Trigger* é um recurso extremamente poderoso em um SGBD, permitindo a execução de um bloco de comandos a partir da inserção, atualização ou remoção de um ou mais registros de uma tabela. Dessa forma é possível, por exemplo, programar

a atualização de um registro em uma tabela a partir da atualização de um registro de outra tabela.

Apesar de muito interessante, essa funcionalidade também é bastante perigosa num cenário de controle de acesso. Imagine, por exemplo, duas tabelas A e B onde B possui uma chave estrangeira para A e a atualização de um registro na tabela A acarrete, através de uma *Trigger*, a atualização de todos os registros de B que apontem para o registro atualizado em A. Imagine também que existe um usuário que tem acesso a todos os registros de A, mas não tem acesso a determinados registros de B, o que se deseja testar é: se o usuário atualiza um registro em A e existe um registro em B onde ele não tem acesso e que aponta para o registro atualizado em A, o registro em B será ou não atualizado pela *Trigger*?

Para testar o comportamento do VPD nesse caso, foi criada uma *Trigger* que faz com que sempre que houver uma atualização do campo *N_COMMENT* da tabela *NATION*, o campo *S_COMMENT* de todos os registros da tabela *SUPPLIER* que apontam para o registro atualizado na tabela *NATION* (*S_NATIONKEY* = *N_NATIONKEY*) sejam atualizados com o mesmo valor. Apesar de não apresentar uma funcionalidade real para o negócio, essa *Trigger* é suficiente para o teste. O código da *Trigger* é apresentado na Figura 56.

```
create or replace TRIGGER "TRIGGER_TESTE"
after update of N_COMMENT on TPCH.NATION for each row
begin
  update TPCH.SUPPLIER
  set S_COMMENT = :new.N_COMMENT
  where S_NATIONKEY = :new.N_NATIONKEY;
end;
```

Figura 56 – *Trigger* de teste do comportamento do VPD com o *framework*

A *Trigger* foi criada pelo usuário *TPCH*, que é o mesmo usuário dono das tabelas do TPC-H, e que não possui nenhum perfil e nem o privilégio *EXEMPT ACCESS POLICY*. O teste foi realizado pela usuária *Alison* cujo perfil (gestora de depósito do Canadá) permite que ela acesse todos os registros da tabela *NATION*, mas limita o seu acesso na tabela *SUPPLIER* aos fornecedores do Canadá. Primeiramente, para verificar se a *Trigger* estava funcionando corretamente, uma atualização na tabela *NATION* foi feita para a nação Canadá. O resultado da atualização é apresentado na Figura 57.

```
SQL> connect alison
Informe a senha:*************
Conectado.
SQL> select s_nationkey, s_name, s_comment from tpch.supplier
  2  where s_nationkey = 3 and s_suppkey < 100;

S_NATIONKEY S_NAME                    S_COMMENT
----------- ------------------------- -------------------------------
          3 Supplier#000000013        ironically busy packages thrash
          3 Supplier#000000020        finally regular asymptotes play
          3 Supplier#000000091        carefully ironic packages impres

SQL> select n_nationkey, n_name, n_comment from tpch.nation
  2  where n_name = 'CANADA';

N_NATIONKEY N_NAME                    N_COMMENT
----------- ------------------------- -------------------------------
          3 CANADA                    foxes among the bold requests

SQL> update tpch.nation
  2  set n_comment = 'Teste Trigger'
  3  where n_name = 'CANADA';

1 linha atualizada.

SQL> select n_nationkey, n_name, n_comment from tpch.nation
  2  where n_name = 'CANADA';

N_NATIONKEY N_NAME                    N_COMMENT
----------- ------------------------- -------------------------------
          3 CANADA                    Teste Trigger

SQL> select s_nationkey, s_name, s_comment from tpch.supplier
  2  where s_nationkey = 3 and s_suppkey < 100;

S_NATIONKEY S_NAME                    S_COMMENT
----------- ------------------------- -------------------------------
          3 Supplier#000000013        Teste Trigger
          3 Supplier#000000020        Teste Trigger
          3 Supplier#000000091        Teste Trigger
```

Figura 57 - Resultado do teste de *Trigger* executado pela usuária *Alison* em registros do Canadá

Ao perceber que a *Trigger* funcionou corretamente, o teste foi repetido para a nação Brasil. O resultado do segundo teste é apresentado na Figura 58.

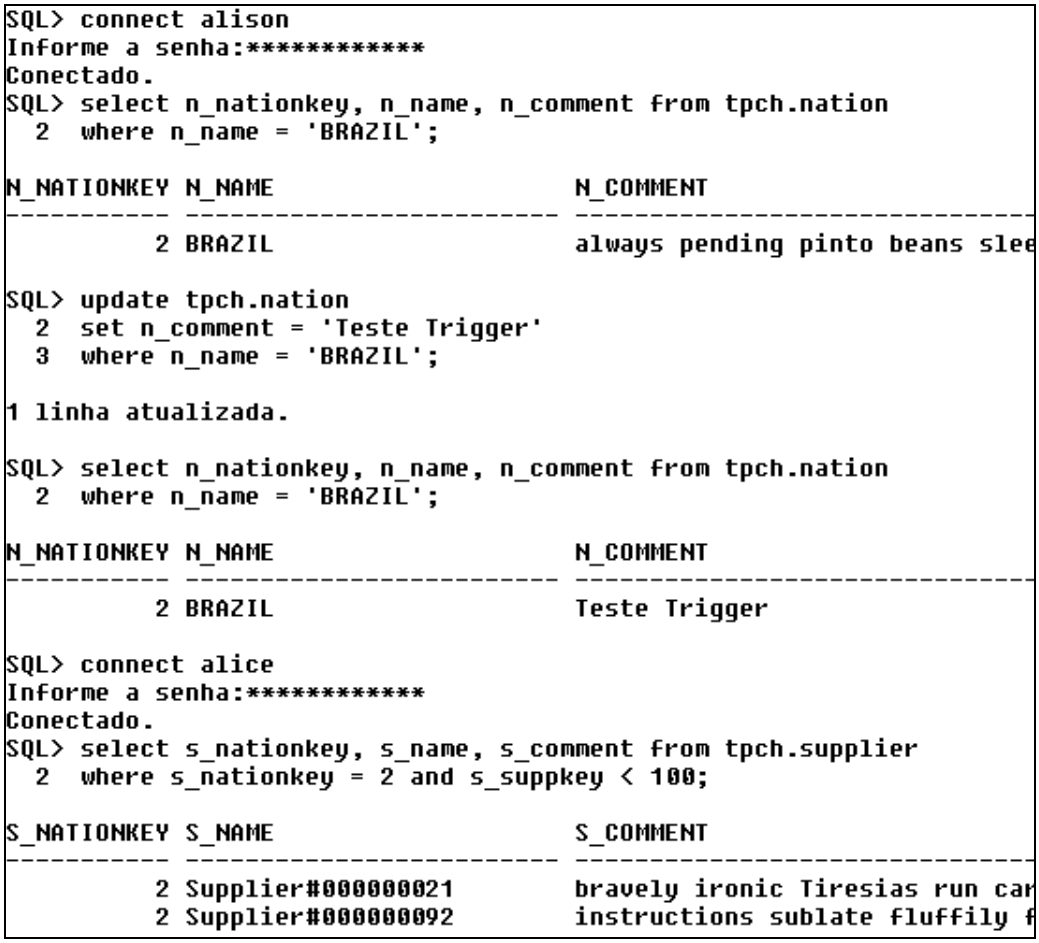

Figura 58 - Resultado do teste de *Trigger* executado pela usuária *Alison* em registros do Brasil

Dessa vez os registros na tabela *SUPPLIER* não foram atualizados, ou seja, o controle de acesso foi garantido, impedindo que a usuária atualizasse um registro que ela não tem acesso. Contudo, *Alison* não é a dona da *Trigger*, mas sim o usuário *TPCH* e, dependendo da política da empresa, pode ser necessário conceder o privilégio *EXEMPT ACCESS POLICY* para esse usuário, pois ele é o dono das tabelas e deve ver todos os dados. Portanto, o teste foi mais uma vez repetido para a nação Brasil, mas agora com o privilégio *EXEMPT ACCESS POLICY* concedido ao usuário *TPCH*. O resultado do terceiro teste é apresentado na Figura 59.

```
SQL> connect alison
Informe a senha:*************
Conectado.
SQL> select n_nationkey, n_name, n_comment from tpch.nation
  2  where n_name = 'BRAZIL';

N_NATIONKEY N_NAME                    N_COMMENT
----------- ------------------------- --------------------------------
          2 BRAZIL                    always pending pinto beans sleep

SQL> update tpch.nation
  2  set n_comment = 'Teste Trigger'
  3  where n_name = 'BRAZIL';

1 linha atualizada.

SQL> select n_nationkey, n_name, n_comment from tpch.nation
  2  where n_name = 'BRAZIL';

N_NATIONKEY N_NAME                    N_COMMENT
----------- ------------------------- --------------------------------
          2 BRAZIL                    Teste Trigger

SQL> connect alice
Informe a senha:*************
Conectado.
SQL> select s_nationkey, s_name, s_comment from tpch.supplier
  2  where s_nationkey = 2 and s_suppkey < 100;

S_NATIONKEY S_NAME                    S_COMMENT
----------- ------------------------- --------------------------------
          2 Supplier#000000021        Teste Trigger
          2 Supplier#000000092        Teste Trigger
```

Figura 59 - Resultado do teste de *Trigger* executado pela usuária *Alison* em registros do Brasil com o dono da *Trigger* com o privilégio *EXEMPT ACCESS POLICY*

Dessa vez os registros na tabela *SUPPLIER* foram atualizados, ou seja, o controle de acesso não foi garantido, permitindo que a usuária atualizasse um registro que ela não tem acesso. Isso acontece, pois o privilégio *EXEMPT ACCESS POLICY* concedido ao usuário *TPCH* faz com que o Oracle ignore as políticas ao executar a *Trigger* permitindo, dessa forma, que o usuário atualize registros que ele não tenha acesso. Para que o usuário *TPCH* possua acesso a todos os dados das tabelas sem comprometer o controle de acesso, é recomendado criar um perfil com o predicado '1=1' e associado a esse usuário. A concessão do privilégio *EXEMPT ACCESS POLICY* deve ser feita com muito cuidado para evitar problemas como o apresentado nesse teste.

Capítulo 6: CONCLUSÃO

Segurança de acesso a dados é uma questão importante para as empresas. Regras de autorização são tradicionalmente implementadas em aplicações, que definem suas próprias políticas de segurança e às aplicam na camada de aplicação. Contudo, se ma regra muda, todas as aplicações que implementam a regra devem ser atualizadas. Portanto, esse é um problema bastante complexo em cenários típicos onde existem sistemas legados.

Abordagens de segurança de dados podem ser classificadas como DAC (*Discricionary Access Control*), MAC (*Mandatory Access Control*), propostos por [DoD 1983], ou RBAC (*Role-Based Access Control*) proposto por [Ferraiolo e Khun, 1992]. Contudo as implementações existentes dessas abordagens são difíceis de gerenciar e necessitam de profissionais qualificados.

Neste trabalho, apresentamos um framework flexível para gestão e execução de regras de autorização em bancos de dados corporativos, também denominado de FARBAC (*Flexible Approach for Role Based Access Control*) [Azevedo, Puntar, *et al.*, 2010]. O framework é composto de dois módulos: (i) Gestão de Regras de Autorização (GRA) e (ii) Execução de Regras de Autorização (ERA). O módulo ERA foi implementado através do *Virtual Private Database* (VPD) do SGBD Oracle, e avaliado através de consultas e dados do *Benchmark* TPC-H para as operações de seleção, inserção, atualização, remoção, *Merge* e execução de *Trigger*. Os resultados compilados na Tabela 2 mostraram a eficácia e viabilidade da proposta.

Tabela 2 – Compilação de resultados da avaliação da proposta

Operação	Observação
Seleção	Para a operação de seleção foram executadas diversas consultas a fim de cobrir padrões comuns de seleção de dados no banco de dados, ou seja, consultas simples, consultas com funções de agregação, consultas aninhadas, consultas com agrupamento, consultas com filtro de

	agrupamento e consultas com *case*. O resultado foi satisfatório para todas as consultas, obtendo-se o resultado esperado em todos os casos. Contudo, observou-se que em algumas consultas o usuário que possui restrição de acesso pode ter uma leitura errada do estado do negócio. Para evitar esse tipo de problema, todos os usuários devem entender muito bem a sua visão dos dados.
Inserção	A operação de inserção está intimamente relacionada ao parâmetro *UPDATE_CHECK* do VPD. Quando o parâmetro é definido com o valor verdadeiro (*true*), o VPD verifica se o usuário terá acesso aos dados que ele está tentando inserir e, se não tiver, a operação não é realizada. Já quando o parâmetro é definido com o valor falso (*false*), o VPD não faz essa verificação, o que permite que o usuário insira dado que ele mesmo não terá acesso depois de inseridos. A definição do valor desse parâmetro fica a cargo da organização de acordo com tipo de controle que deseja realizar.
Atualização	Assim como a operação de inserção, a operação de atualização também está intimamente relacionada ao parâmetro *UPDATE_CHECK* do VPD, contudo, o comportamento, neste caso, é um pouco mais complexo. Como a atualização é feita sobre registros já existentes, ela só será realizada caso o usuário já possua acesso ao registro que deseja atualizar, independentemente do valor definido para o parâmetro *UPDATE_CHECK*. No caso em que o usuário possui acesso ao registro, quando o *UPDATE_CHECK* é definido com o valor verdadeiro (*true*), o VPD verifica se o usuário ainda terá acesso aos dados após a atualização e, se não tiver, a operação não é realizada. Já quando o *UPDATE_CHECK* é definido com o valor falso (*false*), o VPD não faz essa verificação, o que permite que o usuário atualize o registro para um estado que ele mesmo não terá acesso depois de atualizado. Novamente a definição do valor desse parâmetro fica a cargo da

	organização de acordo com tipo de controle que deseja realizar.
Remoção	A remoção é a mais simples das operações, quanto ela é executada e o usuário tem acesso ao registro ele é removido, caso contrário o VPD impede que ele seja removido.

Merge	A operação *Merge* foi a única que não se comportou da forma esperada lançando a seguinte exceção:
	`ORA-28132: A intercação na sintaxe não suporta polítias de segurança.`
	`28132.0000 – "Merge into syntax does not support security policies"`
	`* Cause: Merge into syntax currently does not support a security policy on the destination table.`
	`* Action: use the insert / update DML stmts on the table that has a security policy defined on it.`
	Oracle ainda não suporta a operação de *Merge* em tabelas que possuem controle de acesso. Como não é possível utilizar o merge em conjunto com o VPD, a própria Oracle recomenda que sejam usados comandos *Insert* e *Update* equivalentes para garantir o controle de acesso [Jeloka, 2008].
	Os testes de *Merge* foram realizados nas versões 10.2.0.3 e 11.1.0.

Trigger	A execução de *Trigger* apresentou um comportamento curioso, mas que permite mais um nível de personalização nas políticas de segurança. No teste realizado, uma *Trigger* que atualizava dados que o usuário não tem acesso, a partir da atualização de dados que ele tem acesso, garantia o controle de acesso somente quando o usuário dono da *Trigger* não possuía o privilégio *EXEMPT ACCESS POLICY*. Dessa forma, é possível definir se o controle de acesso deve ser garantido ou não em *Triggers* através da concessão ou não privilégio *EXEMPT ACCESS POLICY*.

Em comparação aos métodos de controle de acesso atualmente utilizados (principalmente em relação ao VPD tradicional), o *framework* apresentou vantagens como a necessidade de apenas uma função de autorização por tabela, a viabilidade de

cadastro de todas as regras no modelo proposto e a possibilidade de reaproveitamento das regras entre perfis e tabelas. Essas vantagens levam a uma facilidade de manutenção por parte do DBA já que, para alterar as regras e perfis, basta alterar as informações das tabelas do modelo de autorização, ou seja, não há necessidade de atualizar o código da função, o que torna a proposta flexível. Além disso, testes de desempenho apresentados em [Puntar *et al*., 2010] mostram a eficiência da proposta em relação ao VPD tradicional.

Os próximos passos sobre o módulo de execução de regras de autorização incluem a extensão do modelo, para possibilitar a definição de hierarquias de perfis com mais de dois níveis, para tratar regras que sejam aplicadas somente a determinados tipos de operação (seleção, inserção, atualização ou remoção), para tratar regras de mascaramento de colunas e também para tratar regras aplicáveis a colunas relevantes. Apesar do VPD possuir funcionalidades que permitam a realização dos três últimos itens, o *framework* não permite a definição de regras diferentes para cada tipo de política, ou seja, se a política for aplicada em um conceito com mascaramento de coluna, todas as regras relacionadas a esse conceito usarão essa funcionalidade.

Deseja-se ainda estudar a possibilidade de definição de regras públicas que representem os dados sem restrição de acesso, estudar uma forma de aplicação de perfis a grupos de usuários e fazer uma avaliação do custo de desempenho em executar o controle de acesso na aplicação ou no banco de dados. Além de todos esses itens, um trabalho futuro essencial é a generalização da proposta, para que ela seja aplicável a qualquer banco de dados, e não apenas ao Oracle.

Os próximos passos sobre o módulo de gestão de regras de autorização incluem pesquisar e desenvolver uma interface que seja simples o suficiente para que usuários do negócio possam cadastrar regras de autorização.

REFERÊNCIAS BIBLIOGRÁFICAS

AZEVEDO, L. G.; PUNTAR, S.; THIAGO, R.; BAIAO, F.; CAPPELLI, C., 2010. *A Flexible Framework for Applying Data Access Authorization Business Rules*. In: 12th International Conference on Enterprise Information Systems, 2010, Funchal, Madeira, Portugal. 12th International Conference on Enterprise Information Systems, 2010. p. 275-280.

BRG, 2000. *Defining Business Rules ~ What Are They Really?*, Rev. 1.3, 2000, http://www.businessrulesgroup.org/first_paper/BRG-whatisBR_3ed.pdf, acessado em abril de 2010.

DoD, 1983. *Trusted Computer Security Evaluation Criteria*. Department of Defense, DoD 5200.28-STD.

DOMINGUES, G. R.; CIFERRI, C. D. A.; CIFERRI, R. R., 2008. *VisualTPCH: Uma Ferramenta para a Geração de Dados Sintéticos para Data Warehouse*. In: Anais da Sessão de Demos do XXIII Simpósio Brasileiro de Banco de Dados, p.31-36, 2008.

EISENBERG, A., MELTON, J., KULKARNI, K., MICHELS, J.E., ZEMKE, F., 2004. *SQL:2003 Has Been Published*. SIGMOD Record, 33(1):119-126, 2004.

FERRAIOLO, D.F. e KHUN, D. R., 1992. *Role-Based Access Control*. In: 15th National Computer Security Conference, pp. 554—563, Baltimore, MD.

GIURI, L. E IGLIO, P., 1997. *Role templates for content-based access control*. In: Proceedings of the Second ACM Workshop on Role-Based Access Control (Fairfax, Virginia, United States, November 06 - 07, 1997). RBAC '97. ACM, New York, NY, 153-159.

JELOKA, S.; MULAGUND, G.; LEWIS N. *et al*., 2008. *Oracle Database Security Guide, Oracle RDBMS 10gR2*. Oracle Corporation. http://download.oracle.com/docs/cd/B19306_01/network.102/b14266.pdf, acessado em abril de 2010.

LEVINGER, J. E.; NEEDHAM, P.; PESATI, V.; TATA, S. *et al*., 2003. *Oracle Label Security Administrator's Guide*. Oracle Corporation. http://download.oracle.com/docs/cd/B14117_01/network.101/b10774.pdf, acessado em abril de 2010.

MURTHY, R.; SEDLAR, E., 2007. *Flexible and efficient access control in oracle*. In ACM SIGMOD 2007 International Conference on Management of Data, pp. 973-980, Beijing, China.

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 (Alexandria, Virginia, USA, October 27 - 31, 2008). CCS '08. ACM, New York, NY, 289-298.

OLSON, L. E.; GUNTER, C. A.; COOK, W. R.; E WINSLETT, M., 2009. *Implementing Reflective Access Control in SQL*. In: Lecture Notes in Computer Science, 2009, Volume 5645/2009, 17-32.

PUNTAR, S.; AZEVEDO, L.; BAIÃO, F.; CAPPELLI, C., 2010. *Testes de desempenho comparando modelo flexível com VPD tradicional*. Relatórios Técnicos do DIA/UNIRIO (RelaTe-DIA), RT-0013/2010, Disponível em: http://www.seer.unirio.br/index.php/monografiasppgi/article/view/1206

RIZVI, S.; MENDELZON, A.; SUDARSHAN, S.; ROY, P., 2004. *Extending query rewriting techniques for fine-grained access control*. In: SIGMOD 04, Paris, France, June.

SOX, 2009. *Sarbanes-Oxley: Financial and Accounting Disclosure Information*. http://www.sarbanes-oxley.com/section.php?level=1&pub_id=SOA-Manual, acessado em abril de 2010.

TPC Council, 2008. *TPC Benchmark H Standard Specification Revision 2.8.0*. Transaction Processing Perfermance Council. http://www.tpc.org/tpch/spec/tpch2.8.0.pdf, acessado em abril de 2010.

YANG, L., 2009. *Teaching database security and auditing*. ACM SIGCSE'09, v.1, issue 1, pp. 241—245, 2009.