Projeto e implementação do framework Outer-tuning: auto sintonia e ontologia para bancos de dados relacionais

Alternative title: Design of the Outer-tuning framework: self-tuning and ontology for relational databases

Rafael Pereira de Oliveira
Pontifícia Universidade Católica do Rio de Janeiro - PUC-Rio
Rio de Janeiro, Brasil
rpoliveira@inf.puc-rio.br

Sérgio Lifschitz
Pontifícia Universidade Católica do Rio de Janeiro - PUC-Rio
Rio de Janeiro, Brasil
sergio@inf.puc-rio.br

Ana Carolina Almeida
Universidade do Estado do Rio de Janeiro
Rio de Janeiro, Brasil
ana.almeida@ime.uerj.br

Edward Hermann Haeusler
Pontifícia Universidade Católica do Rio de Janeiro - PUC-Rio
Rio de Janeiro, Brasil
hermann@inf.puc-rio.br

RESUMO

Neste trabalho discutimos a construção e o projeto de um *framework* - chamado Outer-Tuning - que dá apoio à sintonia fina (semi) automática de sistemas de bancos de dados por meio de uma ontologia específica. São apresentados os aspectos arquiteturais baseados em componentes, projeto de interface e, principalmente, a abordagem adotada para contemplar tanto uma ontologia de domínio como uma ontologia de tarefas em um sistema de informação integrador. Discutimos também aspectos relacionados com a máquina de regras de inferência e questões relacionadas ao uso de linguagens lógicas. Por fim, alguns resultados experimentais permitem uma avaliação preliminar das contribuições esperadas pelo *framework*.

Palavras-Chave

Framework, design, sintonia fina, banco de dados, ontologia

ABSTRACT

In this paper, we discuss the architectural project and construction of a framework called Outer-Tuning, which supports (semi) automatic tuning of database systems through a specific ontology. The architectural aspects component-based, interface design, and especially the approach taken to include both a domain ontology and an ontology tasks in an integrative information system. We also discuss aspects of machine inference rules and issues related to the use of logical languages. Finally, some experimental results allow an assessment of the expected framework contributions.

Categories and Subject Descriptors

H [**Information Systems**]; H.2 [**DATABASE MANAGEMENT**]; H.2.7 [**Database Administration**]

General Terms

Design, experimentation

Keywords

Framework, design, tuning, databases, ontology

1. INTRODUÇÃO

O trabalho de administração e sintonia fina (*tuning*) de bancos de dados é complexo e exige especialização e conhecimentos fundamentais nesta área da computação. Vários sistemas foram desenvolvidos para dar apoio ao DBA (database administrator) nesta atividade, cujo objetivo principal é garantir disponibilidade e bom desempenho. Em particular, alguns sistemas permitem a realização de tarefas de sintonia fina de maneira (semi) automática [20][6]. Em sua maioria, são sistemas desenvolvidos a partir do conhecimento de um DBA especialista, que geram algoritmos e heurísticas de manutenção do sistema de banco de dados.

O principal objetivo da sintonia fina é buscar um melhor desempenho para o banco de dados. Para isso, realizam-se ajustes de suas configurações, parâmetros e projeto físico, seleção de estruturas de acesso, duplicação de estruturas físicas, sempre de acordo com a carga de trabalho executada no banco. Existem diversas propostas de ferramentas

que auxiliam a sintonia fina através da automatização de estratégias para a seleção de índices e visões materializadas, particionamento, reescrita de consultas, entre outras.

Apresenta -se aqui o resultado do projeto de arquitetura de *software*, desenvolvimento e avaliação do *framework* Outer-Tuning [1] ferramenta baseada em ontologia e criada para apoiar DBAs (e possivelmente outros usuários) nas escolhas envolvidas na atividade de sintonia fina. Em particular, discutimos os principais requisitos funcionais, o fluxo operacional e as diversas fases de execução necessárias para atender os requisitos. Também discutimos em detalhes os componentes propostos para a arquitetura, as alternativas de projeto e as escolhas efetuadas, incluindo a interface do usuário. Por estarmos utilizando uma ontologia especificamente construída para apoio à tarefa de sintonia fina em bancos de dados relacionais, discutimos também o uso de linguagens como DATALOG, OWL-DL e SWRL e sua integração na construção e implementação do *framework*.

Para a validação do *framework* foi realizada a instanciação bem como a extensão da ontologia de sintonia fina usada por ele, através da inclusão de duas novas heurísticas. A partir de uma carga de trabalho, essas novas heurísticas realizam a seleção e criação de visões materializadas. Com o uso de um *benchmark*, foi realizada uma avaliação experimental da qualidade dos resultados obtidos pelo Outer-Tuning, através da documentação, avaliação e comparação das ações de sintonia fina inferidas, como resultado de sua execução, por suas heurísticas.

A Seção 2 traz os fundamentos e conceitos envolvidos neste artigo, além de alguns trabalhos relacionados à nossa pesquisa. Logo após, a Seção 3 documenta o nosso método de pesquisa. A Seção 4 apresenta o projeto de arquitetura do *framework* Outer-Tuning e as principais escolhas realizadas durante a sua implementação. Discute-se o uso de uma ontologia para inferir ações de sintonia fina, a integração de uma máquina de regras com o *framework* e os principais componentes de *software* envolvidos. Em seguida, a Seção 5 traz uma avaliação preliminar do Outer-Tuning através da execução de duas heurísticas de sintonia fina automática envolvendo a seleção de visões materializadas. Por fim, a Seção 6 lista as principais conclusões e trabalhos futuros.

2. FUNDAMENTOS E TRABALHOS RELACIONADOS

O trabalho apresentado por [1] trouxe a proposta de um *framework*, chamado Outer-Tuning, para apoiar o trabalho de sintonia fina (semi)automática em sistemas de bancos de dados. A ideia básica da ferramenta Outer-Tuning consiste em oferecer mecanismos que permitam ao DBA uma tomada de decisão mais consistente com relação às possíveis ações de sintonia fina, baseada em alternativas e respectivos custos. Devido ao escopo da pesquisa e limitações de tempo não foi realizado o projeto por completo, consequentemente, não houve implementação do *framework*. Alguns testes preliminares foram realizados por meio de simulações utilizando o *software* Protégé [1].

A pesquisa apresentada nesse artigo tem como ênfase a descrição do projeto de arquitetura do *framework* Outer-Tuning. A partir da ontologia específica de sintonia fina em bancos de dados relacionais, proposta em [1] e estendida em [17], buscamos atender os requisitos - listados na Seção 4.1 – e os desafios de especificação dos componentes e integração com uma máquina de regras. Apesar do *framework* Outer-Tuning ter sido definido usando como base ações de sintonia fina sobre índices, buscamos nesta pesquisa estender a discussão para o uso de visões materializadas (VM), de forma a não particularizar as soluções até então obtidas e permitir uma melhor compreensão das vantagens do uso de uma ontologia em todo o processo. Mais detalhes da ontologia de sintonia fina podem ser encontrados em [17].

O conceito de visão, em bancos de dados relacionais, diz respeito basicamente à uma consulta definida sobre relações do banco de dados. A definição da visão fica armazenada no catálogo e, enquanto um objeto do banco de dados, pode ser utilizado em consultas da mesma forma que as relações que pertencem ao esquema relacional. Já visões materializadas são consultas que têm não somente suas definições como também seus resultados fisicamente armazenados no banco de dados. As visões materializadas são implementadas pelos SGBDs mais populares como PostgreSQL, SQL Server, Oracle, entre outros [9].

Visões materializadas possuem custos. Não é necessário ter apenas espaço em disco suficiente, mas também requer o custo de criação, manutenção e execução de uma consulta SQL utilizando uma VM. Quando uma tabela é atualizada, consequentemente o conteúdo da visão materializada se torna desatualizado. Logo, surge a necessidade de seleção do conjunto de visões materializadas hipotéticas que tragam o maior benefício e os menores custos possíveis, dada uma carga de trabalho.

A formalização da escolha do conjunto de visões materializadas consideradas viáveis é dada pela seguinte definição: "*Dado um esquema de um banco de dados R, armazenado em espaço D e uma carga de trabalho de consultas Q, escolher um conjunto de consultas V sobre R para ser materializado de forma que o somatório do tamanho dos elementos de V seja menor que D*" [9].

Além da questão de viabilidade, deve-se considerar quais VMs podem ser úteis para melhorar o desempenho de um sistema de banco de dados. A escolha do conjunto de consultas, cujo custo de execução da carga de trabalho é minimizado, dado uma restrição de espaço em disco e um custo de manutenção, é um problema com complexidade exponencial [19]. Logo, não é possível materializar todas as possíveis visões hipotéticas a melhorar o desempenho por pelo menos dois motivos: (i) o espaço em disco necessário para materializar todas as opções inviabiliza esta escolha e (ii) o custo de manter as visões materializadas atualizadas [9] é proibitivo.

Especificamente para a seleção automática e semiautomática de VMs, existem ferramentas que utilizam diferentes métodos de seleção do conjunto de consultas para a criação dessas estruturas de acesso. Entre elas, podemos citar: (i) pesquisas que utilizam heurísticas gulosas [24], [2], [8]; (ii) métodos de seleção baseados em heurísticas randômicas [13], [25], [21], [22]; (iii) abordagens baseadas em heurísticas genéticas e evolucionárias [24], [23], [12]; (iv) heurísticas híbridas [25], que são uma mistura de abordagens genética e randômica; e (v) heurísticas específicas [7].

Apesar de utilizarem VMs para realizar sintonia fina, nenhuma das abordagens encontradas se propõe a fornecer ao DBA uma justificativa das decisões e tornar explícito o raciocínio utilizado na atividade de sintonia fina. Também não possuem a possibilidade de comparação ou inclusão de novas

[1] http://protege.stanford.edu/ acesso em 18/02/2015.

heurísticas sem a necessidade de uma alteração do seu código fonte, tornando a comparação e adaptação das estratégias à carga de trabalho difícil e trabalhosa.

Todas estas características são oferecidas pelo *framework* Outer-Tuning. Trata-se de uma abordagem que faz uso de uma ontologia de aplicação que permite a inferência de ações de sintonia fina. A ontologia de domínio descreve os conceitos envolvidos na tarefa de sintonia fina e a ontologia de tarefas contém heurísticas capazes de analisar a carga de trabalho instanciada na ontologia de domínio. Neste artigo discutimos, exatamente, o projeto e arquitetura do *framework* Outer-Tuning, seus componentes e alternativas de arquitetura, com o detalhamento das decisões tomadas para efetiva implementação da ferramenta.

3. MÉTODO DE PESQUISA

Inicialmente, foi realizada uma pesquisa bibliográfica para: (i) investigar os passos necessários para o processo de sintonia fina; (ii) conhecer ferramentas de sintonia fina que realizem a seleção de visões materializadas; (iii) levantar possíveis arquiteturas para um *framework* de sintonia fina; (iv) levantar métodos de desenvolvimento de *frameworks*; e (v) avaliar máquinas de regras passíveis de integração com o *framework* e compatíveis com a ontologia de sintonia fina. Em um segundo momento, foi realizado o planejamento dos passos necessários para a inferência de ações de sintonia fina (Figura 1).

Para o desenvolvimento do projeto de arquitetura do *framework*, buscou-se utilizar a metodologia de projeto dirigido por *hot spot* (*Hot Spot Driven Design*) [18], tendo como base a ontologia de tarefa. Os *hot spots* são as partes flexíveis do *framework*. A essência dessa metodologia é identificar os *hot spots* na estrutura de classes de um domínio e, a partir disto, construir o *framework*.

Por fim, para a avaliação do *framework* foi realizada uma instanciação e extensão da ontologia de sintonia fina, onde foram incluídas duas novas heurísticas para, dada uma carga de trabalho, realizarem a seleção e criação de visões materializadas. Com a utilização de um *benchmark* para gerar a carga de trabalho, o Outer-Tuning foi então executado e teve as ações de sintonia fina inferidas pelas suas heurísticas (resultado de sua execução) documentadas, avaliadas e comparadas.

4. FRAMEWORK OUTER-TUNING

As fases de execução do *framework* foram definidas de acordo com os requisitos funcionais levantados a partir da análise dos requisitos do Outer-Tuning e pela avaliação das ferramentas de seleção de visões materializadas citadas na Seção 2. Após a definição das fases, foi desenvolvido o projeto de arquitetura do *framework*. Também é apresentado na Seção 4.4 uma análise sobre o uso de linguagens lógicas para realizar tarefas de sintonia fina.

4.1 Requisitos funcionais do framework Outer-Tuning

Durante a definição do escopo do *framework* Outer-Tuning, levantou-se requisitos desejáveis. Entre eles, que a partir da ontologia de sintonia fina proposta, fosse capaz de:

(i) Capturar a carga de trabalho de um banco de dados de forma contínua; **(ii)** Extrair da carga de trabalho os conceitos necessários à ontologia de domínio; **(iii)** Instanciar os conceitos extraídos da carga de trabalho na ontologia; **(iv)** Dar a oportunidade ao DBA (ou outro usuário) de escolher qual(is) heurística(s) de sintonia fina deseja executar durante o processo de sintonia fina; **(v)** Executar as heurísticas escolhidas pelo DBA e inferir as ações de sintonia fina previstas nas heurísticas; **(vi)** Exibir ao DBA justificativas semânticas sobre as ações inferidas pelas heurísticas escolhidas; **(vii)** Executar no banco de dados as ações de sintonia fina inferidas pelas heurísticas, de forma automática ou supervisionada pelo DBA. A partir dos requisitos listados, foram definidas as fases de execução do Outer-Tuning.

4.2 Fases de execução do framework Outer-Tuning

O fluxo definido para o Outer-Tuning (Figura 1) se inicia com o monitoramento do banco de dados de forma não intrusiva e contínua (2), que captura a carga de trabalho (3) submetida ao banco de dados (1) por seus usuários. Em seguida são extraídos (4) da carga de trabalho os conceitos, cujos valores são necessários para que as heurísticas contidas na ontologia possam realizar suas estimativas (5). Depois de extraídos, eles são instanciados na máquina de regras (6), usando a ontologia de sintonia fina (7). A partir daí, através das heurísticas de sintonia fina, definidas na ontologia (5), a máquina de regras (6) infere as ações de sintonia fina (8), dada aquela carga de trabalho (3). Caso seja o modo automático (9), serão executadas todas as ações inferidas no banco de dados (1). Já se for um processo semiautomático, as ações de sintonia fina deverão ser exibidas pela interface ao usuário (10), que escolherá (11) quais ações executará no banco de dados (1).

4.3 Projeto de arquitetura baseada em componentes

Todo o *design* de um *framework* é criado a partir de como as funcionalidades vão ser transformadas em código fonte e como elas vão se comunicar entre si. Durante o projeto de arquitetura foi realizado o planejamento dos passos necessários para a inferência de ações de sintonia fina (Figura 1). A partir daí, criou-se o projeto de arquitetura do *framework* Outer-Tuning (Figura 2), que foi seguido de sua implementação.

A arquitetura escolhida para o Outer-Tuning é baseada em componentes. Componente de *software* é uma parte não-trivial, quase independente, e substituível, de um sistema que cumpre uma função clara no contexto da arquitetura [5]. Devido ao caráter experimental da ferramenta e as múltiplas tecnologias envolvidas (ex.: SGBDs, máquinas de regras, ontologia, bibliotecas, mais de uma linguagem de programação), decidiu-se que os componentes poderiam facilitar a comunicação entre as partes e deixar cada uma das fases de execução (Figura 1) independentes, e caso necessário, poderiam ser substituídos/mantidos de forma compartimentalizada sem a propagação de bugs. Além dessas vantagens, componentes são umas das mais viáveis e usadas formas de reuso de *software*, principalmente entre desenvolvedores de *software* livre [15].

Como resultado da escolha de uma arquitetura baseada em componentes, decidiu-se que o Outer-Tuning seria desenvolvido como um *framework* de aplicação, que por definição é uma aplicação semi-completa, construída com uma coleção

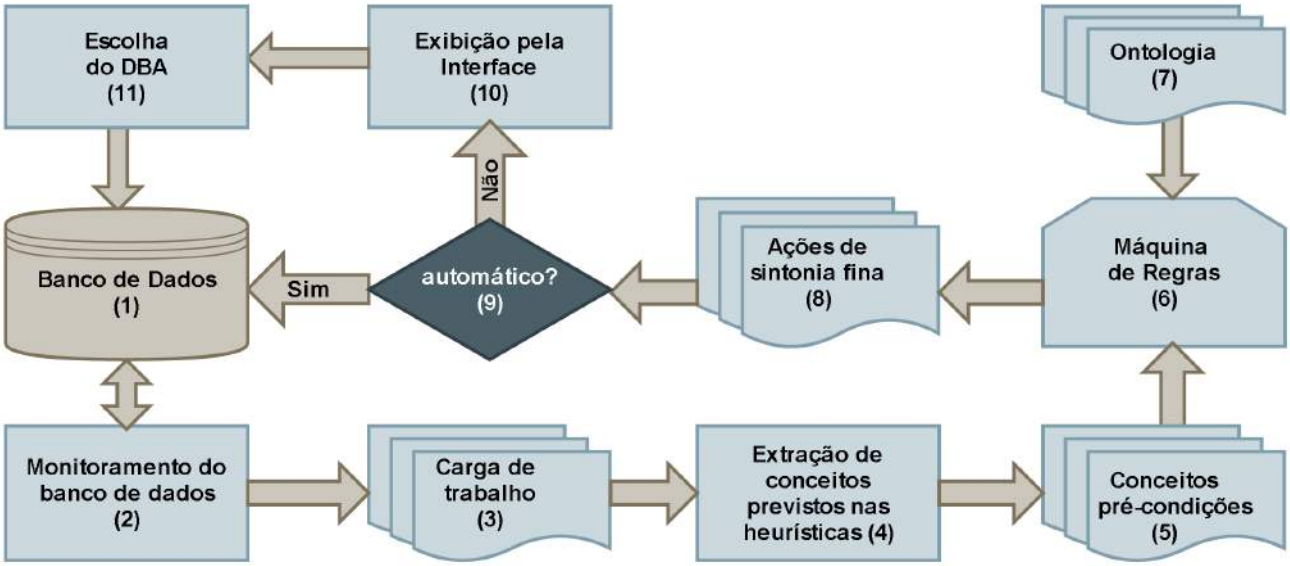

Figura 1: Fluxo de execução projetado para o *framework* Outer-Tuning

organizada de componentes de *software* reusáveis [16]. A escolha desse tipo de *framework*, assim como o uso de componentes, foi feita para dar qualidade ao *software*, possibilitar uma futura evolução para *software* orientado a serviços [3], mas principalmente permitir um baixo acoplamento entre as partes do *software*.

De acordo com o fluxo de execução proposto, o Outer-Tuning teve seus componentes definidos e especializados para cada etapa do ciclo. Na Figura 2, são enumerados e descritos de forma detalhada os principais elementos da arquitetura proposta:

(1) Base: Apesar do desenvolvimento baseado em componentes ser uma das formas de reuso de *software* mais eficientes da engenharia de *software*, foi utilizada em paralelo a esse tipo de reuso uma biblioteca de funções para que os componentes pudessem compartilhar funções ordinárias e redundantes, além de dar acesso a um sistema de log compartilhado.

(2) Ontologia: A ontologia de aplicação contém a ontologia de domínio (define conceitos) e a ontologia de tarefas (define as heurísticas através de regras "se-então"). É a principal parte extensível do *framework*. Durante o projeto, ela foi propositalmente isolada da implementação para que o DBA pudesse inserir/alterar/excluir heurísticas sem ter que alterar componentes do *framework*. Na prática isso dá ao DBA uma independência da implementação, ele não precisa saber como o *framework* foi implementado para, por exemplo, corrigir um modelo de custo de alguma heurística. As regras são definidas em uma linguagem declarativa (SWRL), logo o DBA precisa se preocupar apenas em "o que" a heurística deve fazer, e não "como" ela deve fazer.

(3) Máquina de regras: Uma das decisões importantes tomadas, foi decidir "em qual momento" e "como" a máquina de regras seria utilizada. Uma máquina de regras (ou motor de inferência) é o componente pelo qual as regras "se-então" das heurísticas (definidas na ontologia de tarefas) são selecionadas e executadas. Foi usado um componente do tipo "caixa-preta", onde não é possível o acesso ao código-fonte. A única maneira de adaptar o componente é substituí-lo por outro que possua interfaces compatíveis com o atual. Para a implementação foi utilizada a máquina de regras Jess [2]. Sua escolha se deu pela compatibilidade entre as interfaces e os requisitos. O requisito funcional (iv) (Seção 4.1) determina que o DBA possua uma forma de habilitar e desabilitar as heurísticas antes de iniciar um processo de sintonia fina. A biblioteca Jess possui uma interface que permite habilitar/desabilitar um conjunto de regras SWRL (usadas nas heurísticas), e isso foi um aspecto decisivo para que o DBA, através da interface do Outer-Tuning, pudesse escolher quais heurísticas deveriam ser executadas ou não.

(4) Banco de dados: Este também foi considerado um componente caixa preta. Sua comunicação com o *framework* (5) e (15) se dá através dos drivers de conexão usados pelos componentes *CapturadorCargaDeTrabalho* (6) e *ExecutorDeFuncoes* (7). Vale lembrar que, o termo banco de dados aqui tem o significado de SGBD. Logo, a comunicação pode ser realizada com qualquer banco de dados gerenciado por um determinado SGBD.

(5) Carga de trabalho: A carga de trabalho capturada consiste em comandos SQL do tipo DML (*Data Manipulation Language* - Linguagem de manipulação de dados), os seus respectivos planos de execução e sua frequência. O termo carga de trabalho não abrange os dados manipulados por estes comandos DML.

(6) *CapturadorCargaDeTrabalho*: É o componente responsável por adquirir a carga de trabalho com um intervalo de tempo pré-determinado pelo DBA para realizar a sintonia fina. Para a conexão entre o banco de dados e o *framework* foi projetada uma classe de conexão que utiliza o driver JDBC [3]. Logo, o Outer-Tuning pode suportar os principais SGBDs do mercado que implementam essa tecnologia.

(7) *ExecutorDeFuncoes*: Toda heurística possui um conjunto de conceitos que são pré-condições e que devem ser

[2]http://www.jessrules.com/ acesso em 18/02/2015
[3]http://www.oracle.com/ acesso em 18/02/2015

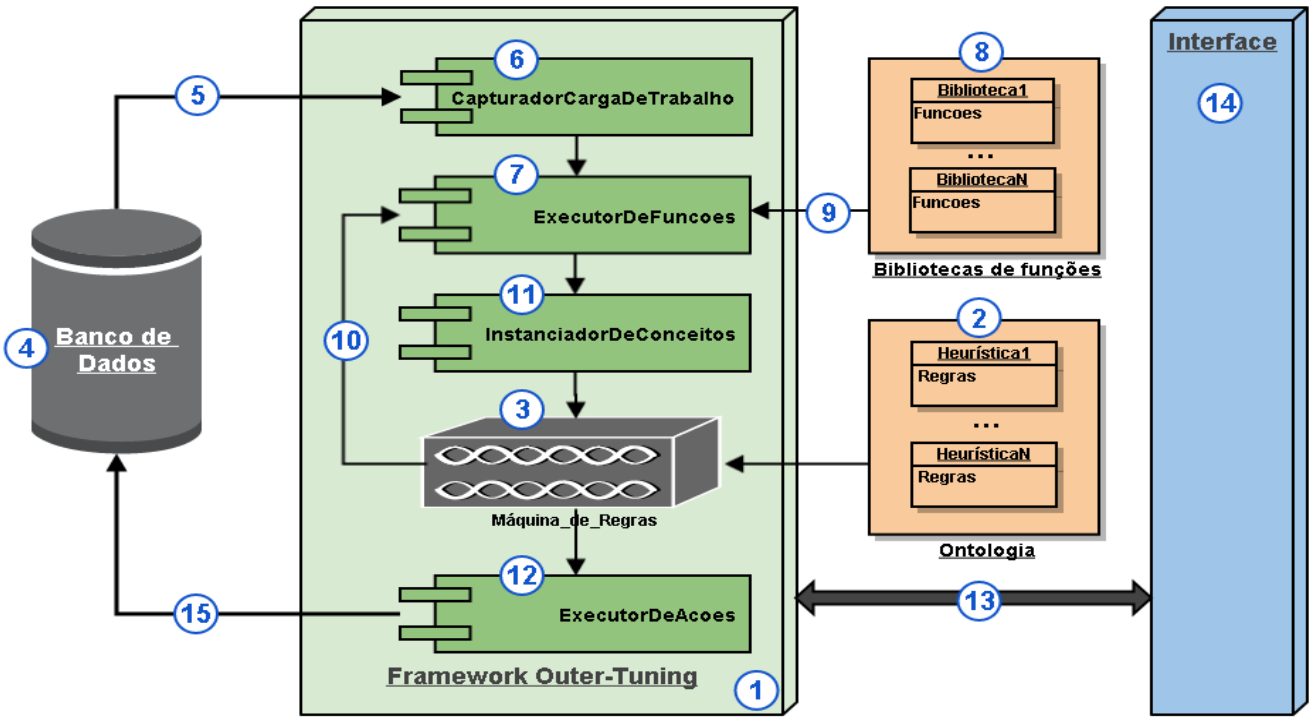

Figura 2: Projeto de Arquitetura do *framework* Outer-Tuning

instanciados na máquina de regras para que haja inferência de ações de sintonia fina. O componente *ExecutorDeFuncoes* é o responsável por extrair informações da carga de trabalho e gerar os indivíduos de conceitos que são pré-condições. Ele possui três fontes de entrada de dados para este componente:

(i) Carga de trabalho capturada pelo componente *CapturadorCargaDeTrabalho* (6); **(ii)** Definição das funções, parâmetros de entrada e saída (10) que são definidas pelas heurísticas escolhidas pelo DBA durante a inicialização da ferramenta, para instanciar os conceitos que são seus pré-requisitos. **(iii)** Implementação das funções (9) obtidas pelas bibliotecas de funções implementadas e compiladas. Essas bibliotecas são fisicamente independentes do *framework* e lidas em tempo de execução pelo componente *ExecutorDeFuncoes* (7). Isso faz com que elas possam ser adicionadas ou substituídas sem nenhuma intervenção no código ou (re)compilação da ferramenta.

Para realizar o seu trabalho, o *ExecutorDeFuncoes* recebe do componente *CapturadorCargaDeTrabalho* (6) a carga de trabalho capturada. Também recebe da máquina de regras (10) a lista de conceitos que devem ser extraídos, junto com as assinaturas das funções que realizaram a extração. De posse das assinaturas das funções, o componente *ExecutorDeFuncoes* lê (9) a biblioteca que contém o código fonte compilado das funções, e as executa. O parâmetro de entrada das funções é sempre originado da carga de trabalho e os parâmetros de saída são sempre indivíduos de conceitos pré-condições das heurísticas. De posse dos indivíduos, resultado da execução das funções, o componente *ExecutorDeFuncoes* envia-os para o componente *InstanciadorDeConceitos*.

(8) Bibliotecas de funções: Sua função é aglutinar bibliotecas de código fonte compiladas responsáveis por extrair conceitos da carga de trabalho. As bibliotecas de funções, assim como a ontologia, são partes extensíveis do *framework* e foram projetadas para trabalhar isoladas da estrutura do *framework*. Elas são lidas e executadas em tempo de execução, podendo ser incluídas, alteradas, ou apagadas sem nenhuma intervenção no código fonte do *framework*.

(9) Comunicação entre bibliotecas e *ExecutorDeFuncoes*: É realizada através de interface definida no componente *ExecutorDeFuncoes* que busca no repositório as funções desejadas.

(10) Conceitos pré-condições das heurísticas: O componente *ExecutorDeFuncoes* recebe da máquina de regras os conceitos que são pré-condições e a assinatura das funções contidas na biblioteca de funções que extraem conceitos.

(11) *InstanciadorDeConceitos*: É responsável por instanciar os indivíduos gerados pela execução das funções pelo *ExecutorDeFuncoes* na máquina de regras. Estes indivíduos são de conceitos que são pré-condições para execução das heurísticas. Foi determinado que esta funcionalidade se tornasse um componente pela complexidade da tarefa. Este componente tem a responsabilidade de instanciar qualquer conceito, atributo ou relacionamento de conceitos definidos na ontologia, independente da versão da ontologia ou heurística escolhida. Ele também é responsável por não instanciar dois indivíduos iguais, visto que as bibliotecas de funções podem extrair conceitos iguais em momentos diferentes. Por exemplo: duas consultas podem compartilhar uma mesma tabela "vendas". Logo, este componente controla indivíduos duplicados durante a instanciação.

(12) *ExecutorDeAcoes*: Monitora a máquina de regras, e captura as ações de sintonia fina inferidas pelas heurís-

ticas. Após a captura, o componente transmite as ações de sintonia fina para a interface do *framework* para que o DBA possa escolher quais ações deseja efetivamente executar. Após a escolha, o componente *ExecutorDeAcoes* recebe uma lista de ações da interface e as executa no banco de dados. Compartilha o sistema de conexão com o componente *CapturadorCargaDeTrabalho* através da *Base* (1).

(13) Comunicação *framework* − interface: Por se tratar de uma ferramenta de sintonia fina, considerou-se importante que a interface fosse desacoplada do *framework*. A independência é justificada pela possibilidade de dar ao DBA a opção de rodar a ferramenta em sistemas operacionais sem interface gráfica (ex.: servidores Linux) que proveem potência computacional, e ao mesmo tempo poder usufruir de uma interface amigável e que possa evoluir sem nenhuma restrição tecnológica em relação ao Outer-Tuning. Para que isso fosse viável, decidiu-se implantar um protocolo de comunicação entre a Interface(14) e o *framework*. O padrão de projeto escolhido foi o quadro-negro (*blackboard*)[11], pela sua facilidade de implementação, e pela possibilidade da execução sem nenhuma interface. Com a utilização do padrão quadro-negro, a interface durante os testes e implementação pôde ser substituída por um arquivo de configuração, que facilita a depuração do *framework* de forma mais rápida e prática por parte do desenvolvedor.

(14) Interface: É responsável pela interação com o DBA. Foi projetada para ser uma interface web e possuir 5 seções: (a) Seleção de heurísticas: possibilita o usuário selecionar quais heurísticas deseja executar; (b) carga de trabalho capturada: é possível visualizar estatísticas e a carga de trabalho executada no banco de dados em tempo real; (c) ações de *tuning*: permite selecionar e executar as ações inferidas pela ontologia no banco de dados; (d) ações hipotéticas: exibe as ações de sintonia fina hipotéticas, que ainda não foram consideradas positivas para a carga de trabalho mas são opções possíveis de sintonia fina; e (e) log: tela de visualização do log de execução do *framework*. Permite ao DBA observar o comportamento da ferramenta e detalhes internos de sua execução.

4.4 O uso de linguagens lógicas para realizar sintonia fina

Linguagens lógicas têm sido usadas com frequência como base para a construção de modelos e ferramentas para representar conhecimento. A Lógica Clássica de Primeira Ordem (LPO) poderia ser uma escolha natural mas é, no entanto, indecidível, ou não possui mecanismo para verificação computacional de consistência de uma base de conhecimento.

Description Logic (DL), por outro lado, representa um fragmento decidível de LPO. O poder de expressão de DL é bastante abrangente, permitindo desde a representação de hierarquias simples até modelos associados a diagramas de classes e/ou comportamento em UML, passando por modelos de Entidades e Relacionamentos. DL é, de certa forma, uma herança de iniciativas como DATALOG, que permitem a especificação e validação de propriedades impossíveis de serem verificadas em LPO. Em DATALOG é possível se operar queries recursivas sem a necessidade de definir transações. Pelas razões teóricas descritas acima, em conjunção com outras razões históricas, DL tornou-se a lógica subjacente na formalização da maior parte das Ontologias que necessitam de algum teste formal de validação.

Existem diversas lógicas de descrição com diferentes capacidades de especificação. A OWL-DL (Ontology Web Language - DL) é a lógica de descrição que foi escolhida para implementar o aspecto dedutivo da Web Semântica. Diversas ontologias na Web são formalizadas em OWL-DL. Trata-se de uma linguagem que tem um poder de expressão bastante significativo, sendo possível a garantia de representação de qualquer modelo de classes em UML e/ou ER, com verificação automática de consistência [4].

Neste nosso trabalho, o uso de regras para representar as heurísticas de sintonia fina em bancos de dados relacionais se mostra mais natural que puramente uma descrição via ontologia das mesmas. Por esta razão, a adição de regras à OWL-DL mostrou-se bastante atrativa. Dentre as diversas possibilidades, optamos por adicionar regras DATALOG, a saber, rulesML à OWL-DL. Na literatura, SWRL (Semantic Web Rules Language) é precisamente o que mais se adequa à nossa necessidade.

O trabalho que descrevemos neste artigo resolve o problema da integração de regras à OWL-DL implementando o mapeamento de regras restritas SWRL em DL [10]. O único cuidado que tomamos que garante a adequação do mapeamento, é que em todas as regras de sintonia fina não há variáveis compartilhadas entre antecedentes de uma regra e consequentes da mesma regra, e no mínimo um nominal é compartilhado entre antecedente e consequente. Desta forma, a nossa implementação de SWRL para sintonia fina de Banco de Dados via ontologias em OWL-DL, apesar do uso de regras, é decidível.

5. AVALIAÇÃO

Faz parte da definição de um *framework*, a capacidade de ser estendido. Como método de avaliação, o Outer-Tuning foi estendido para a inserção de duas novas heurísticas de sintonia fina, ambas responsáveis pela seleção e criação de visões materializadas: a heurística de benefícios [14] e a heurística de expectativas [17]

Devido sua arquitetura, todos os componentes (com exceção dos considerados "caixa-preta") do Outer-Tuning são passíveis de extensão. No caso de inserção/alteração/exclusão de heurísticas de sintonia fina no Outer-Tuning, são necessárias apenas alterações no arquivo da ontologia, e se necessário, inclusão de novas funções para extração de conceitos nas bibliotecas de funções (veja (2) e (8) Figura 2). Essa característica tornou o processo de inclusão de duas novas heurísticas de VMs um processo rápido. Foram necessárias apenas o estudo das heurísticas e a formalização de suas regras de inferência na linguagem SWRL. Não foi necessário alteração de código fonte ou conhecimento da implementação. Todas as alterações se mantiveram em alto nível, com manipulação apenas da ontologia.

As heurísticas foram executadas e tiveram seus resultados avaliados e comparados através do Outer-Tuning. Para a geração da carga de trabalho durante os testes foi utilizado o *benchmark* TPC-H que possui uma carga de trabalho analítica (OLAP), propícia para a avaliação de ferramentas de seleção de visões materializadas. A carga de trabalho foi utilizada com o SGBD PostgreSQL na versão 9.1 e sistema operacional Ubuntu 12.04 32bits. O hardware usado foi um Quad-Core 1.6Ghz, 4GB de RAM e 100GB de disco rígido.

Foram considerados dois tipos de VMs durante a análise dos resultados, as benéficas e as maléficas. Uma VM foi considerada benéfica quando as consultas reescritas e forçadas

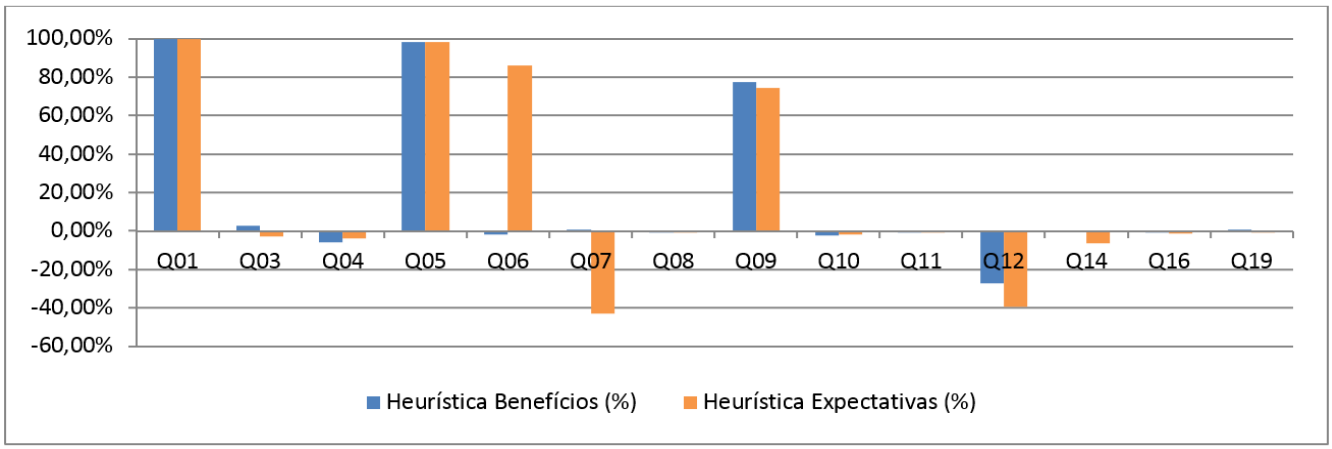

Figura 3: Ganho (em %) no tempo de execução das consultas na presença de visões materializadas sugeridas pelas heurísticas de benefícios e de expectativas

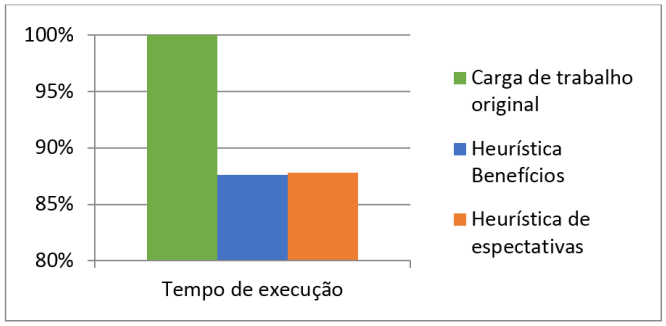

Figura 4: Custo total de cada heurística em relação á carga de trabalho original

a utilizarem a VM trouxeram um ganho para a execução da carga de trabalho, e maléficas quando trouxeram perdas para a execução da carga de trabalho.Uma VM maléfica ocorre quando a reescrita de consulta remove parâmetros de restrição e torna a VM maior que a quantidade de páginas lidas na consulta original. Mais detalhes sobre VMs maléficas em [17].

As duas heurísticas inferiram visões materializadas para a carga de trabalho utilizada. A heurística de benefícios apontou três visões benéficas para a carga de trabalho (Q01, Q05, Q09) e duas maléficas (Q4, Q12). Já a heurística de expectativa apontou quatro visões benéficas (Q01, Q05, Q06, Q09) e três maléficas (Q3, Q04, Q07, Q12, Q14) (Figura 3).

Ambas as heurísticas (Figura 4) trouxeram um ganho positivo equivalente para a carga de trabalho (12,4% e 12,2%). Porém o Outer-Tuning evidenciou que a heurística de benefícios teve o custo de criação e armazenamento de VMs menor que a heurística de expectativas. Enquanto a heurística de benefícios teve 5 VMs criadas, a de expectativas teve 9 VMs criadas.

Pôde-se comprovar, através do Outer-Tuning, que uma mesma heurística pode trazer benefícios ou prejuízos de acordo com a carga de trabalho utilizada. Entendeu-se que a qualidade das ações de sintonia fina está relacionada diretamente com a qualidade da heurística e com a compatibilidade entre a heurística e a carga de trabalho. Independentemente do resultado apresentado por ambas heurísticas, o teste mostra que o *framework* é capaz de trabalhar simultaneamente com mais de uma heurística, é capaz de comparar as soluções apresentadas e a inclusão de novas heurísticas é feita em alto nível através da ontologia.

Com o uso do Outer-Tuning, o DBA possui informações suficientes para avaliar quais heurísticas são interessantes para a sua carga de trabalho. Ele também consegue utilizar sua experiência para adequar as heurísticas através da alteração das mesmas na ontologia, e da inclusão de novas heurísticas. Nada impede, por exemplo, que o DBA realize uma junção entre as heurísticas de benefícios e expectativas criando a sua própria heurística capaz de selecionar as VMs positivas que ambas inferiram, mas evitando criar as VMs negativas (Q7, Q3, Q14) como fez a heurística de expectativas.

Foi mostrado pelos testes que o Outer-Tuning é capaz de: i) inferir ações de sintonia fina úteis; ii) comparar heurísticas de sintonia fina inseridas na ontologia; e iii) apoiar o DBA na tarefa de sintonia fina ao fornecer informações relevantes para que ele possa adequar as heurísticas à carga de trabalho ou inserir novas heurísticas na ontologia de sintonia fina.

Para que o leitor possa conhecer detalhes do funcionamento, visualizar as informações que são disponibilizadas ao DBA e a dinâmica do processo de sintonia fina, está disponível no endereço *http://outertuning.biobd.inf.puc-rio.br* uma versão demonstrativa.

6. CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho foi apresentado o fluxo de controle e a arquitetura usada para desenvolver uma ferramenta de apoio à sintonia fina de banco de dados. Essa ferramenta foi projetada de forma a possibilitar a sua extensão em alto nível, sem exigir que o DBA precise demandar tempo para entender o código-fonte. Além disso, foi usado um *benchmark* conhecido pela comunidade de banco de dados e foram aplicadas duas heurísticas diferentes para validar que a ferramenta consegue alternar entre heurísticas de forma

amigável, ou seja, em alto nível, usando apenas a ontologia e componentes independentes da forma de implementação. Dessa forma, foi possível comparar os resultados de ambas heurísticas, mesmo que estes se mostrassem com ganhos parecidos. O objetivo do presente trabalho não é demonstrar que uma heurística é ou não melhor do que outra, mas sim possibilitar a flexibilidade ao DBA na escolha, comparação e extensibilidade de uso de heurísticas diferentes. E durante essa comparação, obter justificativas semânticas para as decisões tomadas pelas heurísticas descritas na ontologia usada pela ferramenta.

Como trabalho futuro se pretende elaborar um template para padronização e melhoria da apresentação das justificativas semânticas e apresentar a ontologia de forma gráfica para que o DBA possa manipular os conceitos que ele precisa indicar como necessários ao inserir uma nova heurística. Além disso, encontrar uma maneira gráfica de demonstrar toda a evolução do raciocínio da ferramenta para a tomada de determinada decisão.

Adicionalmente, nossa ferramenta se limitou a cargas de dados estáticas do tipo OLAP (*On-line Analytical Processing*) com o objetivo de manipular e analisar um grande volume de dados sob múltiplas perspectivas, e não considerou o custo de manutenção das visões materializadas durante o processo de seleção das ações de sintonia fina. Contudo, somente será necessária a adequação das heurísticas e da ontologia para que o Outer-Tuning possa considerar os custos de atualização dos dados durante o processo de *tuning*. Logo, como trabalho futuro, esperamos remover essa limitação e selecionar ações de sintonia fina para outros tipos de carga de trabalho.

7. REFERÊNCIAS

[1] A. C. B. d. Almeida. *Framework para apoiar a sintonia fina de banco de dados*. PhD thesis, Pontifícia Universidade Católica do Rio de Janeiro - PUC-RIO, 2013.

[2] K. Aouiche, P.-E. Jouve, and J. Darmont. Clustering-based materialized view selection in data warehouses. In *10th E.E.Conf. on Advances in Databases and Information Systems*, ADBIS'06, pages 81–95, Berlin, Heidelberg, 2006. Springer-Verlag.

[3] L. G. Azevedo, F. A. Baião, and F. Santoro. Identificação de Serviços a partir da Modelagem de Processos de Negócio. *V Simpósio Brasileiro de Sistemas de Informação*, pages 133–144, 2009.

[4] D. Berardi, D. Calvanese, and G. De Giacomo. Reasoning on UML class diagrams. *Artificial Intelligence*, 168:70–118, 2005.

[5] A. W. Brown and K. C. Wallnau. Engineering of Component Based Systems. In *Component-Based Software Engineering*, pages 7–15, 1996.

[6] N. Bruno. *Automated Physical Database Design and Tuning*. CRC Press, New York, New York, USA, 2012.

[7] A. W. Carvalho. Criação Automática de Visões Materializadas em SGBDs Relacionais. Master's thesis, Pontifícia Universidade Católica do Rio de Janeiro - PUC-RIO, 2011.

[8] G. Chan, Q. Li, and L. Feng. Optimized design of materialized views in a real-life data warehousing environment. *International Journal of Information Technology*, 7(1):30–54, 2001.

[9] R. Chirkova and J. Yang. Materialized views. *Found. and Trends in Databases*, pages 295–405, 2012.

[10] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML, 2015.

[11] R. Khosla, N. Ichalkaranje, and L. C. Jain. *Design of Intelligent Multi-Agent Systems: Human-Centredness, Architectures, Learning and Adaptation*. Studies in Fuzziness and Soft Computing. Springer, 2004.

[12] M. Lawrence. Multiobjective genetic algorithms for materialized view selection in OLAP data warehouses. In *GECCO '06*, page 699. ACM Press, 2006.

[13] X. Li, X. Qian, J. Jiang, and Z. Wang. Shuffled Frog Leaping Algorithm for Materialized Views Selection. *Intl. Workshop Education Technology and Computer Science*, pages 7–10, 2010.

[14] E. M. T. Morelli and S. Lifschitz. Recriação Automática de Índices em um SGBD Relacional. Master's thesis, Pontifícia Universidade Católica do Rio de Janeiro - PUC-RIO, 2006.

[15] L. D. C. Nascimento and F. Santoro. Análise de interações nas Comunidades Virtuais de Software Livre. *V Simpósio Brasileiro de Sistemas de Informação*, pages 12–23, 2009.

[16] J. L. D. Oliveira, L. Fernando, B. Loja, S. Larissa, V. Vicente, and G. Neto. Um Componente para Gerência de Processos de Negócio em Sistemas de Informação. *VII Simpósio Brasileiro de Sistemas de Informação*, pages 250–261, 2011.

[17] R. P. d. Oliveira. Sintonia fina baseada em ontologias: o caso das visões materializadas. Master's thesis, Pontifícia Universidade Católica do Rio de Janeiro - PUC-RIO, 2015.

[18] W. Pree. Hot-spot-driven framework development. In *Summer School on Reusable Architectures in Object-Oriented software Development*, pages 123–127. ACM, 1995.

[19] R. Ramarkrishnan and J. Gehrke. *Sistemas de gerênciamento de banco de dados*. McGraw Hill, 2008.

[20] D. Shasha and P. Bonnet. *Database Tuning: Principles, Experiments, and Troubleshooting Techniques*. Elsevier Science, 2002.

[21] X. Song and L. Gao. An ant colony based algorithm for optimal selection of materialized view. In *Intelligent Computing and Integrated Systems (ICISS)*, pages 534–536, Oct 2010.

[22] X. Sun and Z. Wang. An Efficient Materialized Views Selection Algorithm Based on PSO. In *Intl. Intelligent Systems and Applications*, pages 1–4. Ieee, May 2009.

[23] S. H. Talebian and S. A. Kareem. A lexicographic ordering genetic algorithm for solving multi-objective view selection problem. In *Conf. on Comp. Research and Development*, ICCRD '10, pages 110–115, 2010.

[24] T. Vijay Kumar and S. Kumar. Materialized view selection using genetic algorithm. In *Contemporary Computing*, volume 306, pages 225–237. Springer Berlin Heidelberg, 2012.

[25] Z. Yuhang, L. Qi, and Y. Wei. Materialized view selection algorithm cssa vsp. In *Computational Intelligence and Natural Computing Proceedings (CINC)*, volume 1, pages 68–71, Sept 2010.