Pontifícia Universidade Católica
do Rio de Janeiro

Rafael Pereira de Oliveira

Sintonia fina baseada em ontologia: o caso de visões materializadas

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática da PUC–Rio

Orientador: Prof. Sérgio Lifschitz

Rio de Janeiro
Março de 2015

Pontifícia Universidade Católica
do Rio de Janeiro

Rafael Pereira de Oliveira

Sintonia fina baseada em ontologia: o caso de visões materializadas

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática do Centro Técnico Científico da PUC–Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Sérgio Lifschitz
Orientador
Departamento de Informática — PUC–Rio

Prof. Daniel Schwabe
Departamento de Informática — PUC–Rio

Prof. José Maria Monteiro Filho
Departamento de Computação — UFC

Prof. Fabio Andre Machado Porto
Laboratório Nacional de Computação Científica — LNCC

Prof. José Eugenio Leal
Coordenador Setorial do Centro Técnico Científico —
PUC–Rio

Rio de Janeiro, 25 de Março de 2015

Rafael Pereira de Oliveira

Graduado em Sistemas de Informação (2011) pelo Centro Universitário de Anápolis - UniEvangélica, também possui graduação em Pedagogia (2007) pela Universidade Estadual de Goiás - UEG. Atua principalmente nos seguintes temas: banco de dados, sintonia-fina de banco de dados, engenharia de software, desenvolvimento de software baseado em componentes, reuso de software e programação.

Ficha Catalográfica

Oliveira, Rafael Pereira de

Sintonia fina baseada em ontologia: o caso de visões materializadas / Rafael Pereira de Oliveira; orientador: Prof. Sérgio Lifschitz. — Rio de Janeiro : PUC–Rio, Departamento de Informática, 2015.

v., 101 f: il. ; 29,7 cm

1. Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui referências bibliográficas.

1. Informática – Tese. 2. banco de dados. 3. sintonia fina. 4. ontologia. 5. framework. 6. visões materializadas. I. Lifschitz, Sérgio. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Agradecimentos

A Deus.

A minha esposa, companheira de jornada, incentivadora e alicerce.

Ao meu orientador que como um verdadeiro mestre me mostrou o caminho, motivou a melhorar, apontou as falhas sempre de forma construtiva, deu o suporte necessário para a pesquisa, e foi um grande amigo.

A minha mãe Maria Lucia, que me alfabetizou e me ensinou que a única forma de vencer é estudando. A meu pai Nilton Gomes pelas bênçãos e confiança que sempre depositou em mim. A meu irmão Tiago pelo amor fraterno e amizade eterna. A meu padrinho Geraldo Donizetti pelo incentivo verdadeiro.

Aos meus amigos do grupo "Sexta-feira Santa" que por seis meses, durante todos os finais de semana, compartilharam comigo a sala 415 e me mostraram a diferença entre aluno e estudante.

A Bruno Olivieri (Mô), André Moreira (Coelhinho) e Igor Vasconcelos (Lampião) pelo apoio, amizade, companheirismo e discussões ferrenhas para decidir se os algoritmos eram O(log N).

A Ana Carolina, que acompanhou esta pesquisa de perto e me ajudou durante toda a caminhada. A Ema Molina, una buena amiga. Los cubanos Liester, Julio Omar y Alain. Otávio Freitas, pelas ótimas conversas. Antony Seabra por todas as consultas e conselhos. Patrick Sava, pelas dicas de programação. A todos os outros companheiros de laboratório que me ajudaram e dividiram este ambiente tão importante na vida de um estudante.

Aos integrantes do Projeto Portinari que tive a oportunidade de conviver e aprender. Professor João Candido pela confiança e ajuda. Sarah, Vera, Fátima, Reinaldo, Isabel, Elisa e Maria Edina pelas grandes amizades. A Noélia, Rose, e Suely por todas as ajudas, palavras de incentivo e carinho. A Juliana e Gabriela, duas amigas muito importantes e que carregarei no coração. A Maria Duarte, uma amiga atenciosa e sempre presente. A Hanna Montana, pelo carinho e conversas. A Eliza Seoud pelas palavras sinceras em um momento decisivo.

A Nega, por me acolher no Rio de Janeiro. A Julio Oliveira e Thiago Nunes pela fraternidade e almoços de domingo.

A PUC-Rio e a Capes, pelo apoio financeiro.

Resumo

Oliveira, Rafael Pereira de; Lifschitz, Sérgio. **Sintonia fina baseada em ontologia: o caso de visões materializadas**. Rio de Janeiro, 2015. 101p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

O *framework* Outer-Tuning serve para apoiar a sintonia fina de índices (automática ou não) em um sistema de banco de dados. Trata-se de uma abordagem que oferece transparência acerca das alternativas disponíveis para possíveis cenários de sintonia fina, possibilitando combinar estratégias independentes para obter um melhor desempenho do SGBD e permitindo a discussão de justificativas para as ações realizadas. Através do uso de uma ontologia específica para sintonia fina de bancos de dados relacionais, é possível adicionar semântica ao processo com o entendimento dos conceitos envolvidos e gerar, de maneira (semi)automática, novas práticas de sintonia fina, que podem ser inferidas a partir das práticas existentes ou de novas regras e conceitos que venham a surgir no futuro. Este trabalho de pesquisa apresenta como contribuição inicial o projeto e implementação do *framework* Outer-Tuning por meio da formalização de uma arquitetura de software que atende aos requisitos funcionais especificados. Este trabalho também contribui com a extensão da ontologia de domínio e a inclusão de novas heurísticas na ontologia de tarefas para contemplar soluções de sintonia fina com o uso de visões materializadas. Desta forma, passa a ser possível propor o uso de heurísticas para realizar a sintonia fina tanto para índices como também para visões materializadas.

Palavras–chave

banco de dados; sintonia fina; ontologia; framework; visões materializadas;

Abstract

Oliveira, Rafael Pereira de; Lifschitz, Sérgio (advisor). **Ontology-based database tuning: the case of materialized views**. Rio de Janeiro, 2015. 101p. MSc.Dissertation — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

The Outer-tuning framework may be used to support automatic (or not) database tuning, particularity index. It is an approach that offers transparency about the available alternatives to feasible tuning scenarios, making it possible to combine either independent strategies or allow discussion of justifications for actions performed in order to obtain better performances. Using a specific ontology for fine tuning relational databases, we add semantics to the process with the understanding of the concepts involved and generate (semi)automatic new tuning actions, which can be inferred from existing practices or new rules and concepts that arise in the future. This research presents as an initial contribution the actual design and implementation of the Outer-tuning framework through the formalization of a software architecture that meets the specified functional requirements. This work also contributes with the extension of the domain ontology and the inclusion of new heuristics to a task ontology, in order to accomplish fine tuning solutions with the use of materialized views. Thus, it becomes possible to propose the use of tuning heuristics for indexes as well as for materialized views.

Keywords

database; database tuning; ontology; framework; materialized view;

Sumário

Lista de figuras

Lista de tabelas

Lista de Abreviaturas

DBA Database Administrator - Administrador de banco de dados

DDL Data Definition Language - Linguagem de Definição de Dados

DML Data Manipulation Language - Linguagem de manipulação de dados

HB Heurística de Benefícios

HE Heurística de Expectativas

HVMH Heurística de Criação de Visões Materializadas Hipotéticas

OWL Web Ontology Language

RDF Resource Description Framework

SGBD Sistema Gerenciador de Banco de Dados

SWRL Semantic Web Rule Language

VM Visão Materializada

VMH Visão Materializada Hipotética (ou candidata)

VMR Visão Materializada Real

1
Introdução

O *framework* Outer-Tuning é uma ferramenta para a realização da atividade de sintonia fina capaz de propor e executar ações sobre o sistema de banco de dados através de uma abordagem que oferece transparência e confiabilidade acerca das alternativas disponíveis para possíveis cenários no SGBD. Possibilita a combinação de estratégias independentes para solucionar um único problema e permite a extração de justificativas para possíveis ações de sintonia fina. Através do uso de uma ontologia específica para sintonia fina de bancos de dados relacionais, é possível entender os conceitos envolvidos e gerar, de maneira automática, novas práticas de sintonia fina, que podem ser inferidas a partir das práticas existentes ou de novas regras e conceitos que venham a surgir no futuro.

1.1
Motivação

Encontrar uma configuração ótima do projeto físico de banco de dados para uma carga de trabalho de forma manual não é uma tarefa trivial. A sintonia fina de banco de dados tem se tornado uma atividade extremamente complexa para as aplicações atuais. É necessário um profundo conhecimento acerca dos detalhes de implementação dos sistemas gerenciadores de banco de dados (SGBDs) e da carga de trabalho submetida (Monteiro08).

Faz parte do trabalho de um DBA descrever todas ou, se inviável, a maioria das soluções disponíveis e avaliadas em uma atividade de sintonia fina. Além da complexidade na seleção das soluções de sintonia fina, o DBA enfrenta o desafio de justificar as decisões tomadas para melhorar o desempenho da execução de uma determinada carga de trabalho. Não há uma ferramenta integrada que o auxilie com argumentos suficientes para justificar as soluções escolhidas pelas ferramentas de sintonia fina, sejam automáticas ou semiautomáticas. Entretanto, um software que realize a sintonia fina de banco de dados, que apresente as alternativas de sintonia que podem ser aplicadas para solucionar um determinado problema e, ao mesmo tempo, fundamente a escolha de cada uma dessas alternativas, pode trazer uma maior confiança em relação ao uso de ferramentas automáticas, e facilitando o trabalho do DBA.

A ontologia de sintonia fina proposta na tese de doutorado de Ana Carolina Brito de Almeida (Almeida13), aborda apenas a seleção de estruturas de acesso do

tipo índices. Além da ontologia, a tese apresentou também a proposta original e o modelo conceitual do *framework* Outer-Tuning .

1.2 Objetivos e escopo dessa dissertação

Este trabalho de pesquisa se apresenta como uma continuação desta tese de doutorado (Almeida13) que foi usada como ponto de partida.

Nesta dissertação de mestrado podemos mencionar como contribuições próprias: (i) a extensão da ontologia de sintonia fina para realizar a tarefa de seleção e criação de visões materializadas; (ii) um passo-a-passo de como realizar a extensão da ontologia de domínio e da ontologia de tarefas, (iii) o projeto completo de arquitetura do *framework* Outer-Tuning; (iv) implementação do *framework* Outer-Tuning e de uma interface web para visualização dos resultados; e (v) uma avaliação experimental do *framework*.

Estas contribuições podem ser detalhadas como mostrado a seguir:

- Extensão da ontologia de domínio, com a inclusão de novos conceitos, atributos e relações envolvidos na tarefa de seleção de visões materializadas;
- Extensão da ontologia de tarefas, com a inclusão de três heurísticas de sintonia fina para seleção de visões materializadas;
- Documentação de um possível fluxo de execução para ferramentas de seleção de visões materializadas;
- Desenvolvimento do projeto arquitetural, implementação e documentação do *framework* Outer-Tuning;
- Uma avaliação experimental da qualidade dos resultados obtidos a partir da execução do Outer-Tuning para execução de sintonia fina utilizando o *benchmark* de referência TPC-H (TPCC15).

A Seção 1.3, define a organização dos capítulos desta dissertação.

1.3 Estrutura da Dissertação

O Capítulo 2 apresenta as definições dos conceitos necessários para o entendimento desta dissertação e do contexto em que está inserida. O Capítulo 3 lista os trabalhos relacionados, as principais heurísticas encontradas para a seleção e criação de visões materializadas, e a pesquisa da tese de doutorado (Almeida13) que propõe uma ontologia para inferir ações de sintonia fina para bancos de dados.

O Capítulo 4 traz a extensão da ontologia de sintonia fina para a seleção e criação de visões materializadas, o Capítulo 5 apresenta o projeto de arquitetura

proposto para o *framework* Outer-Tuning, o Capítulo 6 relata os resultados experimentais do Outer-Tuning e uma avaliação dos resultados. Por fim, o Capítulo 7 apresenta as conclusões, as restrições e os trabalhos futuros desta dissertação.

2
Conceitos básicos

Apresentam-se aqui os principais conceitos e fundamentos envolvidos nesta dissertação de mestrado, necessários para a definição do contexto em que a pesquisa está inserida e suas principais contribuições.

2.1
Ontologia

Na área de computação define-se ontologia como sendo um conjunto de primitivas representacionais que modelam um domínio de conhecimento ou discurso. As primitivas são tipicamente classes (ou conjuntos de objetos), atributos (ou propriedades) e relacionamentos (ou relações entre membros de classes) (Gruber09).

Ontologias podem ser aplicadas com os seguintes objetivos: (i) compartilhar um entendimento comum da estrutura de informação entre pessoas ou agentes de software; (ii) permitir o reuso de elementos do domínio de conhecimento; (iii) tornar explícitas as premissas e suposições do domínio; (iv) separar conhecimentos de domínio do conhecimento operacional; e (v) analisar o conhecimento do domínio (Noy04).

Existe uma classificação para ontologias baseada em sua generalidade, que define quatro tipos básicos (Guarino98):

- **Ontologia de fundamentação**: trata de conceitos gerais que são independentes de um problema particular ou domínio específico. Esse tipo de ontologia pode ser reutilizado na elaboração de novas ontologias.
- **Ontologia de domínio**: descreve classes de conceitos e os relacionamentos entre esses conceitos que definem um domínio específico, basicamente uma área de aplicação.
- **Ontologia de tarefas**: descreve conceitos e relacionamentos entre esses conceitos usados para a realização de uma tarefa/atividade genérica.
- **Ontologia de aplicação**: descreve conceitos que dependem tanto de um domínio em particular, quanto de uma tarefa específica. Ela pode ser uma especialização da ontologia de domínio ou tarefa, mas é específica por ser utilizada dentro de aplicações.

Para o desenvolvimento dessa pesquisa, foi considerada a utilização de uma ontologia de aplicação. Ela possui duas outras ontologias em sua formação:

uma ontologia de domínio, que descreve os conceitos, classes e relacionamentos envolvidos na tarefa de sintonia fina; e uma ontologia de tarefas, que define como cada heurística de sintonia fina deve ser executada, considerando os referidos conceitos.

2.2 OWL

OWL (*Web Ontology Language*) é uma linguagem para definição e instanciação de ontologias WEB. Uma ontologia definida em OWL pode incluir descrições de classes, propriedades e suas instâncias. Dada uma ontologia, OWL consegue através de semântica formal especificar como obter consequências lógicas, que são fatos não literalmente presentes na ontologia mas implícitos por semântica (W3C14).

OWL possui três sub-linguagens onde cada uma é uma extensão da anterior (W3C14):

- **OWL Lite**: Dá suporte à classificação hierárquica e às restrições simples. Embora suporte cardinalidade, ela só permite valores booleanos: 0 ou 1. É a forma mais simples de implementação e com menor expressividade e complexidade quando comparada aos outros tipos de OWL (W3C14).
- **OWL DL**: Para aqueles usuários que querem a máxima expressividade sem perder completitude computacional (possui garantia de que todas as implicações serão computadas) e decidibilidade (todas as implicações terminarão em tempo finito). OWL DL inclui todas as construções da linguagem OWL mas com restrições.
- **OWL FULL**: é dedicada a usuários que precisam de máxima expressividade e a liberdade sintática do RDF (*Resource Description Framework*) sem garantia computacional. Por exemplo: em OWL FULL uma classe pode ser tratada simultaneamente como uma coleção de indivíduos ou um indivíduo. OWL Full permite a uma ontologia aumentar o vocabulário de conhecimento pré-definido de RDF ou OWL.

O OWL utilizado para esta pesquisa, será o OWL DL, que possui a máxima expressividade possível, sem perder a completitude computacional.

2.3 SWRL

Semantic Web Rule Language (SWRL) é uma linguagem de regras criada para especificar regras de transformações de dados que define como sintetizar novos fatos a partir da base de conhecimento armazenada em uma ontologia (Breitman07).

SWRL é baseada em uma combinação de OWL DL e OWL Lite. Isso aumenta o conjunto de axiomas da linguagem OWL para poder incluir cláusulas de *Horn*, o que provê um alto nível de abstração. Isso permite que usuários escrevam regras para raciocínio sobre indivíduos OWL e infiram novos conhecimentos sobre esses indivíduos (W3C14).

As regras propostas possuem a forma de uma implicação entre antecedente(s) (corpo) e consequência(s) (cabeça). O significado pode ser lido como: sempre que a(s) condição(ões) antecedente(s) for(em) verdadeira(s), a(s) consequência(s) será(ão) considerada(s) verdadeira(s). Na linguagem SWRL, antecedente e consequência podem ter zero ou mais átomos onde múltiplos átomos são tratados como uma conjunção. Um antecedente vazio é tratado como uma verdade trivial (satisfeita para todas as interpretações) o que traz uma interpretação verdadeira para todas as consequências. Um consequente vazio, é tratado como trivialmente falso (não satisfeito para qualquer interpretação), o que significa que o antecedente também não satisfará em nenhuma interpretação (W3C14).

2.4 Máquina de regras

Uma máquina de regras é o componente pelo qual as regras de inferência são selecionadas e executadas. As regras são armazenadas em uma base, o que possibilita modificar a base de conhecimento sem precisar recompilar a máquina de regras (Fuks11).

A máquina de regras é o componente responsável por realizar as inferências, onde a partir de um conjunto de dados de entrada (cenário corrente), são procuradas regras que contenham tais dados na parte condicional da regra. A cada regra processada, o fato inferido é adicionado ao conjunto de fatos conhecidos. Para que a regra seja aprovada e infira algum fato, suas premissas devem ser satisfeitas e todas as condições devem ser verdadeiras (Fuks11).

2.5 Framework

Um *framework* é uma aplicação reusável que pode ser especializada para produzir aplicações customizadas. Diferentemente de outras técnicas de reuso orientadas a objetos baseadas em bibliotecas de classes, os *frameworks* têm por

objetivo de reutilizar unidades de software divididas de acordo com os requisitos do software final. Também pode ser considerado um *design* reusável de um sistema que descreve como a aplicação é decomposta em um conjunto de objetos interativos (Fayad99).

Um *framework* pode representar desde uma aplicação completa até um subsistema de uma aplicação. Esse escopo varia de acordo com a possibilidade de reuso das unidades de software que o compõe. Para viabilizar o reuso, o *framework* deve descrever todos os componentes que o compõe, a responsabilidade de cada componente, o modelo de comunicação entre eles, a interface de comunicação para cada componente e um fluxo de controle entre os mesmos (Fayad99).

O mais importante na arquitetura de um *framework* é a divisão dos componentes. Todo o *design* de um *framework* é criado a partir de como as funcionalidades vão ser transformadas em componentes e como eles vão acessar as interfaces dos demais. Apesar de um *framework* também utilizar reuso de código, ele é menos importante que o reuso das interfaces internas que devem ser priorizadas durante o projeto de arquitetura (Fayad99).

Os principais benefícios de um *framework* derivam da modularidade, reusabilidade, extensibilidade, e inversão de controle que ele provem para desenvolvedores, como descrito a seguir (Fayad99):

Modularidade: *Frameworks* se beneficiam da modularidade encapsulando detalhes de implementação voláteis através de interfaces estáveis. Isso ajuda a aumentar a qualidade do software por minimizar o impacto de mudanças na implementação do código. Também reduz o esforço requerido para entender e utilizar os componentes.

Reusabilidade: Interfaces estáveis provém ganhos de usabilidade por definir componentes genéricos que podem ser reutilizados em novas aplicações. Essa reusabilidade concentra o conhecimento do domínio e prioriza o esforço e experiência dos desenvolvedores, evitando que eles tenham o trabalho de recriar e revalidar soluções comuns, dedicando-se ao que realmente é desafiador no desenvolvimento do software.

Extensibilidade: Um *framework* provê métodos explícitos de extensão para interfaces e componentes. Com isso, permite-se alterar o comportamento do *framework* de acordo com o domínio em que está inserido, e garantir a possiblidade de utilização em ambientes que exigem customização das funcionalidades disponibilizadas pelo *framework*.

Inversão de controle: A inversão de controle consiste na capacidade do *framework* em determinar qual conjunto de componentes ou métodos invocar de acordo com eventos externos. A arquitetura do *framework* permite processar diferentes objetos de acordo com um comportamento pré-estabelecido para cada

tipo de entrada fornecida pelo ambiente externo. Dessa forma, ele pode executar componentes ou métodos diferentes para entradas diferentes, se adaptando melhor ao meio que está inserido.

2.6 Modelagem física do banco de dados

A modelagem conceitual de banco de dados é a fase do projeto de banco de dados onde é gerado o esquema conceitual, que é independente do modelo lógico a ser adotado. No esquema conceitual é registrado uma descrição em alto nível dos dados a serem armazenados, juntamente com algumas restrições conhecidas para serem aplicadas. Já a modelagem lógica já possui um nível de abstração mais próximo da implementação em um SGBD e tem como resultado o esquema lógico do banco de dados. Nesta dissertação o foco é dado ao modelo lógico relacional. O esquema, produto da modelagem, define relações (tabelas), atributos e seus domínios, e as restrições de integridade estruturas e semânticas.

Durante a modelagem física do banco de dados realiza-se, entre outros, o mapeamento do modelo lógico nas estruturas físicas oferecidas pelo SGBD escolhido. O esquema físico é específico para cada SGBD, onde se consideram as tecnologias disponíveis e quais benefícios cada uma pode trazer (Date2004).

Durante a modelagem do banco de dados, normalmente ainda não se conhece em detalhes qual a carga de trabalho o sistema de banco de dados receberá. De maneira geral, podemos considerar uma carga de trabalho composta por (i) uma lista de consultas com sua frequência, (ii) uma lista das atualizações (inserções, modificações e exclusões) nas instâncias de dados e suas frequências (Ramarkrishnan08).

Após o desenvolvimento do projeto físico e a obtenção da carga de trabalho real, o esquema físico pode ser refinado de acordo com os padrões da carga de trabalho e as características do SGBD, para que se possa obter o melhor desempenho possível (Ramarkrishnan08). Para essa atividade dá-se o nome de sintonia fina (*tuning*).

2.7 Sintonia fina

A atividade de sintonia fina compreende revisar ou ajustar o esquema físico do banco de dados, parâmetros e configurações do SGBD para ter-se uma maior vazão de dados (*throughput*) e um menor tempo de resposta para os comandos executados (Shasha02). DBAs realizam ajustes no projeto físico do banco de dados através da seleção de estruturas de acesso (índices ou visões materializadas), duplicação de estruturas físicas, determinação dos objetos a serem particionados e seus respectivos

tipos de particionamento, sempre de acordo com a carga de trabalho executada (Salles04)(Ramarkrishnan08).

Existem propostas de ferramentas para sintonia fina que realizam seleção de índices e visões materializadas, particionamento automático, reescrita de consultas SQL, ajustes nas configurações do SGBD (por exemplo, controle de *buffer*), otimização de planos de execução, entre outras (Bruno2012).

2.8 Índices

Um índice em um SGBD é uma estrutura de dados que organiza registros de dados de maneira persistente para otimizar determinados tipos de operações de acesso aos dados. Um índice permite acessar de forma eficiente todos os registros que satisfaçam condições de pesquisa nos atributos de chave de busca do índice (Ramarkrishnan08).

As estruturas de índices fornecem um caminho eficiente para localizar e acessar os dados de maneira mais rápida, além de reduzirem a sobrecarga de busca a um pequeno conjunto de tuplas. Uma chave de busca de um índice é uma sequência de atributos. O índice define um mapeamento entre os valores desta sequência de atributos e seus registros correspondentes (Monteiro08).

Existem duas principais formas de organizar entradas de dados de um índice. A primeira é aplicar uma função hash sobre as chaves de pesquisa das entradas de dados. Já a outra é criar uma estrutura de dados em árvore (normalmente uma Árvore B+) que direcione uma pesquisa por esses dados. (Ramarkrishnan08).

Durante esta pesquisa, foi simulado a existência de estruturas de acesso para verificar como seria o desempenho do banco de dados se existissem. Entre elas estão os índices que foram chamados de índices hipotéticos. Um índice é considerado hipotético quando ele possui apenas informações de metadados, ou seja, não existe fisicamente na base de dados e também não pode ser usado efetivamente como estrutura para acessar o dado hipoteticamente indexado (Monteiro08), (Bruno2012).

2.9 Duplicação de estruturas físicas

Uma alternativa de *tuning* ainda pouco explorada é a duplicação de estruturas físicas, denominadas “clusterização alternativa de dados”. A duplicação pode ser realizada em estruturas como tabelas, por exemplo, para permitir duas ou mais ordenações físicas distintas. Os SGBDs permitem que uma determinada tabela seja ordenada (classificada) por um único critério, que irá determinar a ordem com que os dados serão gravados no disco. Normalmente este critério é a chave primária da tabela, mas pode ser alterado para qualquer outro(s) atributo(s) da tabela. Por

exemplo: uma tabela "*produtos*" pode ter os dados armazenados no disco seguindo a ordem da "*cor*", ou ainda pode ser ordenado pelo "*valor*" do produto (Monteiro08).

Esta ordenação física dos registros que compõe a tabela influencia fortemente o desempenho de determinadas consultas (Monteiro08). A consulta (a) da Tabela 2.1 teria um tempo de execução menor se a tabela "*produtos*" estivesse ordenada pelo atributo "*cor*" em comparação com a mesma tabela ordenada pelo atributo "*valor*".

(a)	(b)
Select count(*) as qtd from produtos group by cor	Select count(*) as qtd from produtos group by valor

Tabela 2.1: Exemplos de consultas beneficiadas por ordenações distintas

Da mesma forma, a consulta (b) teria um desempenho melhor que a consulta (a) se a ordenação da tabela fosse realizada pelo atributo "*valor*", em comparação à ordenação da tabela pelo atributo "*cor*".

A duplicação de estruturas físicas é considerada uma técnica de sintonia fina porque permite ao DBA ter duas ou mais ordenações físicas para a mesma estrutura. Considerando o exemplo anterior, ele poderia duplicar a tabela "*produtos*" e ordená-las, uma pelo atributo cor, outra pelo atributo valor. Dessa forma as duas consultas teriam seus tempos de execução minimizados através da ordenação física. Obviamente, esta técnica eleva o custo de armazenamento, já que as estruturas serão duplicadas a cada nova ordenação física desejada pelo DBA. Também existe o custo de manutenção dos dados, já que qualquer alteração nos registros da tabela "*produtos*" deverá ser replicado em todas as cópias da tabela. No entanto, a duplicação destas estruturas aumenta a disponibilidade dos dados e pode ser uma boa alternativa para ambientes onde haja uma alta frequência de leitura e baixa manutenção dos dados.

2.10 Visões materializadas

Uma visão materializada (VM) é uma visão cujo resultado é calculado e armazenado para uso posterior. Uma visão materializada pode ser usada exatamente como uma tabela na definição de novas consultas ou visões. Esta técnica pode trazer um ganho significativo para a consulta que a utiliza, pois a consulta não precisa recalcular o seu resultado durante a execução (Ramarkrishnan08)(Silberschatz2012).

A manutenção de uma visão materializada incorre em custos. Não é necessário ter apenas espaço em disco suficiente mas, também, há o custo de criação

e manutenção. Quando as tuplas de uma tabela são atualizadas o conteúdo da visão materializada se torna desatualizado (Chirkova11).

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 a uma restrição de espaço em disco e um custo de manutenção, é um problema com complexidade exponencial (Ramarkrishnan08). Não é possível materializar todas as possíveis visões por pelo menos dois motivos: (i) o espaço em disco necessário para materializar todas as opções pode inviabilizar esta escolha e (ii) o custo de manter as visões materializadas atualizadas pode ser proibitivo (Chirkova11).

Assim como os índices hipotéticos, foram consideradas durante esta pesquisa a existência de visões materializadas hipotéticas. Visão materializada é considerada hipotético quando possui apenas informações de metadados, ou seja, não existe fisicamente na base de dados e também não pode ser usada efetivamente para a execução de comandos SQL (Carvalho11)(Monteiro08).

2.10.1 Custos envolvidos no processo de seleção de visões materializadas

Visões materializadas consomem recursos computacionais. Durante o processo de seleção de VMs opta-se por limitar ou otimizar um determinado recurso, de acordo com os objetivos do DBA. Esses recursos são os seguintes:

Custo de armazenamento

De acordo com a capacidade de armazenamento da máquina que mantém o SGBD, o DBA necessita tratar o custo de armazenamento como um objetivo de minimização ou restrição durante a seleção do conjunto de VMs possíveis. Entre as unidades de medidas propostas para esse recurso, é possível utilizar 1) a quantidade de tuplas necessárias para armazenar o conjunto de VMs proposto; e 2) o espaço do disco (em *bits*) necessário para persistir o conjunto de VMs candidatas. Nesta dissertação foi utilizado o espaço em disco como uma restrição de custo através de um parâmetro que determinada o quanto espaço em disco estava disponível para a criação de visões materializadas.

Custo de manutenção

Trata-se do custo de manutenção limite de um conjunto de visões, dado uma frequência de atualização dos dados persistidos. Sua importância principal é garantir que o custo de manutenção do conjunto solução não exceda o ganho proporcionado pelas visões materializadas por esse conjunto. Esse é um fator que está diretamente

ligado ao domínio do banco de dados. Para minimizar esse custo deve-se evitar sempre a criação de VMs que sofrem muita atualização.

Uma visão materializada está atualizada quando a deixamos consistente com as alterações feitas nas relações subjacentes (Ramarkrishnan08). Existem duas perguntas que devem ser feitas durante o processo de atualização de uma visão materializada (Ramarkrishnan08):

- Como atualizamos uma visão quando uma relação subjacente é atualizada?
- Quando devemos atualizar uma visão em resposta a uma alteração em uma dada tabela subjacente?

Manutenção imediata: Uma estratégia possível para a atualização de visões materializadas é recalculá-la quando uma relação subjacente for modificada. Nesse modelo, imediatamente após sofrer uma atualização, a visão sofre um recálculo de seus valores (Ramarkrishnan08).

Manutenção incremental: Quando possível, os algoritmos para manutenção de visões devem ser incrementais, no sentido de que o custo da atualização seja proporcional à alteração, ao invés de arcar com o custo de recalcular a visão por completo. Lembrando que, em uma visão, tuplas das relações subjacentes podem se repetir, visto que as duplicatas não são eliminadas de uma cláusula SQL a menos que seja explicitamente definido que o seja (Ramarkrishnan08).

A principal ideia por trás de algoritmos incrementais é calcular eficientemente as alterações feitas nas tuplas da visão, sejam novas tuplas ou alterações na contagem associada a uma tupla, e se a contagem da tupla tem o valor 0, ela é excluída (Ramarkrishnan08).

Custo de execução de consultas

O objetivo é minimizar o tempo de execução das consultas de uma carga de trabalho utilizando visões materializadas para responder consultas que normalmente seriam respondidas por tabelas do banco de dados.

Para calcular o ganho de tempo na execução dessas consultas observa-se a quantidade de consultas beneficiadas por cada visão e, para cada uma dessas consultas, é feita uma análise do potencial de economia em seu uso. Esse potencial varia de acordo com as características da consulta executada. Por exemplo, consultas com um alto grau de agregação e/ou sumarização são consideradas boas alternativas para se tornarem visões materializadas. Isso porque, quanto maior a agregação e sumarização de uma consulta, menor será o tamanho do resultado que será armazenado no disco. Com isso, as consultas que utilizarão a visão, além de não ter que recalcular o resultado da consulta, terão que ler um conjunto menor

de dados no disco e, consequentemente, o custo de execução é menor do que uma consulta que utilize as tabelas base do banco de dados.

Quando se refere ao grau de agregação e agrupamento, há três cenários possíveis: (1) A visão possui uma única tabela base na sua composição, e pode ou não agrupar esses dados em poucos registros sumarizados; (2) A visão é composta por um conjunto de tabelas (normalmente pares de tabelas) com um padrão de junções entre elas definido, que também podem ou não agrupar os dados; e (3) a visão é composta por um conjunto de tabelas com um padrão de junções indefinido (Chirkova11).

2.10.2 Restrições para o espaço de pesquisa

Para tentarmos restringir o espaço de pesquisa para a seleção de visões (dado que ele é exponencial no número de colunas possíveis para a visão), propõe-se o seguinte (Chirkova11):

- **Capacidade de reescrita da consulta pelo otimizador**: Como pré-requisito para o problema de seleção de visão nós precisamos saber se, e quando, uma consulta pode ser reescrita em termos de visões materializadas. No entanto, esta reescrita está longe do óbvio e algumas consultas não podem ser reescritas de forma eficiente para que usem as visões materializadas propostas pelos algoritmos. Portanto, deve-se prever apenas visões que o otimizador de consulta consegue utilizar ao reescrever a consulta executada no banco de dados.
- **Limite prático do custo de percorrer o espaço de pesquisa**: Para que ocorra o processo de automatização da escolha de visões materializadas, o tempo de execução não pode ser longo. Também, deve-se prever o tempo de reescrita da função. Como o processo de reescrita acontece em tempo de execução, reescritas devem ser feitas em tempo hábil de acordo com o uso da aplicação. Para tal, deve-se admitir apenas consultas de um certo tipo, para que o número de visões possíveis para reescrita seja polinomial no tamanho da entrada no pior caso, e torne a tarefa menos custosa para o otimizador.

2.11 Heurística de benefícios

A heurística de benefícios (Costa2003) propõe a criação de estruturas de acesso no banco de dados (ex.: índices, VMs) atribuindo-lhes benefícios à medida que contribuam de forma positiva nos comandos executados pelo SGBD. Isto é conhecido uma vez que obtém-se o custo de cada comando e quando este custo

poderia cair, caso existissem as estruturas de acesso hipotéticas. Quando o valor do benefício de uma estrutura hipotética se torna tão grande quanto o seu custo de criação, disparam-se ações para transformá-las em estruturas reais (Morelli06). Analogamente, a carga de benefícios de uma estrutura de acesso será reduzida, caso esta potencialmente aumente o custo de um comando.

A heurística de benefícios já foi utilizada em outras pesquisas de sintonia fina (Salles04), (Morelli06), (Monteiro08), (Carvalho11), (Almeida13). Sua principal ideia é atribuir benefícios às estruturas de acesso hipotéticas cada vez que se estime que uma consulta otimize o desempenho se a estrutura de acesso hipotética existisse fisicamente no banco de dados. A efetiva criação da estrutura de acesso é determinada quando o benefício acumulado chega a um limite definido. Este limite é o custo de criação física da estrutura de acesso no banco de dados.

A definição da heurística de benefícios também trouxe os conceitos benefício e malefício para a discussão de sintonia fina. O benefício de uma estrutura de acesso é incrementado quando o otimizador ou a ferramenta de sintonia fina decide que a estrutura de acesso é útil para o comando avaliado caso esteja presente no banco de dados. Já o malefício é incrementado quando a consulta avaliada não se beneficiou com a estrutura de acesso.

2.12 Conclusões

Neste capítulo foram apresentados os conceitos e fundamentos necessários para contextualizar a pesquisa apresentada. Foram apresentados conceitos envolvidos com o uso de ontologias, a linguagem OWL e máquinas de regras. Para contextualizar a proposta de arquitetura, foi apresentado o conceito de *framework* e suas características. Foi apresentada também a atividade de sintonia fina, as estruturas de acesso índices e visões materializadas, seguida da heurística de benefícios.

O Capítulo 3 a seguir descreve pesquisas de sintonia fina realizadas no laboratório BioBD e outras encontradas na literatura. Também apresenta a proposta do uso de ontologias para propor ações de sintonia fina.

3
Sintonia fina e ontologia

Descrevem-se aqui pesquisas sobre sintonia fina de bancos de dados e ontologias. A seção 3 trata de um resumo das pesquisas anteriores e que serviram como base para a realização deste trabalho. Já a seção 3.2 realiza uma revisão bibliográfica sobre seleção de visões materializadas.

3.1
Pesquisas do grupo BioBD em sintonia fina

O grupo de pesquisa BioBD da PUC-Rio realiza pesquisas em sintonia fina automática e semiautomática de bancos de dados relacionais desde 2002 sob a orientação do professor Sérgio Lifschitz. A ênfase tem sido dada para estruturas de acesso, como índices e visões materializadas, e também técnicas e ferramentas de engenharia de software na construção de sistemas, como agentes para sintonia fina, uso de ontologias específicas, entre outros.

Marcos Antônio Vaz Salles apresentou em 2004 (Salles04) duas arquiteturas que permitem automatizar a sintonia de índices, onde também foi apresentada pela primeira vez a heurística de benefícios. A independência de intervenção humana é obtida através do uso de agentes de software. A combinação de agentes com SGBDs torna os sistemas autônomos e capazes da auto-sintonia. O autor implementou uma das arquiteturas propostas no código fonte do SGBD PostgreSQL na versão 7 e obteve resultados experimentais com uma carga transacional (TPCC15). Após os testes, ele apresentou uma análise dos resultados onde mostrou os benefícios trazidos com a nova abordagem automática para criação de índices. Cabe observar que as primeiras ideias envolvendo a heurística utilizada neste trabalho foram sugeridas por Rogério Luís de Carvalho Costa (Costa05).

Anolan Yamilé Milanés (Milanes15) propôs uma arquitetura para sintonia fina automática de SGBDs. Nesta pesquisa também foi discutido o uso de agentes de software para a tarefa de sintonia fina, como uma abordagem flexível para a inclusão de atividades de sintonia automática em um SGBD.

Dando continuidade à pesquisa de criação de índices (Salles04), Eduardo Maria Terra Morelli apresentou em (Morelli06) um mecanismo automático de criação, exclusão e reconstruções de índices. Morelli primeiro submeteu à implementação de Salles uma carga de trabalho TPC-H (TPCC15) e depois realizou eliminações e reconstruções de índices automáticas, levando em consideração níveis de preenchimento de páginas alternativos. Também foram realizadas na pesquisa de

Morelli comparações utilizando ferramentas comerciais, Oracle 10g e SQL Server 2005, para avaliar quão eficaz comportou-se a implementação proposta por Salles. Morelli também criou um protótipo capaz de sugerir novos índices e eliminar os que deixaram de ser interessantes, porém, antes da eliminação, o protótipo realiza uma avaliação para verificar se a reconstrução (*reindex*) não seria mais adequada. Morelli criou, inclusive, uma heurística que avalia um índice a ser eliminado e recomenda sua reconstrução, caso atenda a determinados requisitos.

Ainda na linha de sintonia fina utilizando índices, José Maria da Silva Monteiro Filho apresentou em 2008 (Monteiro08) uma nova abordagem não-intrusiva para a manutenção automática e *on-the-fly* do projeto físico de bancos de dados. A abordagem proposta é desacoplada do código do SGBD, pode ser utilizada com qualquer SGBD (diferentemente das pesquisas de Morelli e Salles) e executada sem intervenção humana. A estratégia adotada por Monteiro baseia-se em heurísticas que executam continuamente e, sempre que necessário, modificam o projeto físico corrente, reagindo a alterações na carga de trabalho. Como resultados experimentais, Monteiro apresentou uma instância da abordagem apresentada para solucionar dois importantes problemas relacionados ao projeto físico: a manutenção automática de índices e de *clusters* alternativos de dados.

Andréa Weberling Carvalho apresentou em 2011 (Carvalho11) uma dissertação que propunha a criação automática de visões materializadas em SGBDs relacionais através da arquitetura de auto-sintonia não-intrusiva proposta por (Monteiro08). Seu trabalho se tratou de uma extensão da ferramenta que realizava a criação automática de índices para a criação automática de visões materializadas.

Como trabalho mais recente a esta dissertação, foi apresentada em 2013 por Ana Carolina Brito de Almeida (Almeida13) a proposta de ontologia para a sintonia fina (automática ou não) que proporciona uma abordagem formal para decisões e inferências relativas à atividade de sintonia fina, especificamente à seleção, criação e manutenção de índices. O diferencial dessa abordagem foi oferecer, através de uma ontologia, transparência e confiabilidade acerca das alternativas disponíveis para possíveis cenários no SGBD, por meio de justificativas para as decisões que foram tomadas. Almeida também apresentou a proposta do *framework* denominado Outer-Tuning para execução das heurísticas de sintonia fina através da ontologia apresentada. Dado que é introdução principal para esta dissertação, vamos descrever melhor na seção 5.

3.2
Visões materializadas e sintonia fina

O problema de seleção de visões materializadas consiste em, dada uma carga de trabalho, selecionar o conjunto de consultas para a criação de visões materializadas que tragam o maior benefício (aumento da vazão e/ou diminuição do tempo de resposta) para uma determinada carga de trabalho (Kumar09). Existem propostas de diferentes métodos para a seleção de visões materializadas na literatura. Os mais comuns, são métodos baseados em heurísticas, que podem ser divididas em: gulosas, randômicas, genéticas (ou evolucionárias) e específicas.

Um método de seleção de visões materializadas é considerado guloso quando em cada iteração seleciona-se para criação a VM com maior benefício. Esta seleção de VMs em cada passo continua até que um número pré-definido de VMs seja selecionado ou o espaço disponível para materialização esteja esgotado. Os trabalhos de (Chan01) (Aouiche06), (Kumar09)(Kumar12) são exemplos de uso das heurísticas gulosas. São apresentados algoritmos que consideram o espaço em disco usado para materialização, e a frequência de execução da consulta dentro de uma carga de trabalho. Estes trabalhos sempre escolhem as visões que possuem o maior benefício, considerando o tamanho em disco ocupado e quantidade de consultas respondidas da carga de trabalho.

Heurísticas randômicas (ou aleatórias) também já foram usadas para a seleção de visões materializadas. Elas consideram cada solução como sendo um estado em um espaço de soluções com um custo associado. Entre as estratégias genéricas de otimização já utilizadas para a seleção e visões materializadas, estão: hill-climbing (Phuboon07), (Kumar13), Simulated annealing (Derakhshan06)(Derakshan08), ant colony optimization (colônia de formigas) (Gao10), Particle Swarm Optimization (população de partículas) (Sun09), shuffled frog leaping (Li10a). Apesar destas heurísticas randômicas possuírem estratégias diferentes, todas elas apresentam algoritmos que durante sua execução, randomizam caminhadas em um espaço de estados via uma série de movimentos que são usados para construir arestas entre diferentes soluções no espaço de soluções.

Na literatura também foram encontradas heurísticas genéticas (ou evolucionárias) para a solução do problema de seleção de visões materializadas como (Lawrence06)(Talebian09)(Talebian10)(Yhang10)(Kumar12). Um algoritmo genético (AG) usa uma técnica evolucionária para escolher as visões materializadas. Ela é usada para resolver problemas complexos envolvendo a identificação de um conjunto de soluções que possuem um grande espaço de pesquisa, como é o caso das visões materializadas.

Durante a sua execução, heurísticas genéticas mantem uma população de estruturas, denominadas indivíduos ou cromossomos indivíduos (neste caso visões

materializadas), que comportam-se de forma semelhante à evolução biológica. São realizadas recombinações e mutações dos cromossomos para gerar novos indivíduos e novas possibilidades de solução. Cada indivíduo recebe uma avaliação que é uma quantificação da sua qualidade como solução do problema em questão. No final, a heurística genética seleciona os indivíduos (visões materializadas hipotéticas) com maior benefício de acordo com a carga de trabalho para fazerem parte do conjunto resposta, e serem efetivamente materializados.

3.3 Ontologia de sintonia fina

Foi proposta em 2013 na tese de doutorado de Ana Carolina Brito de Almeida (Almeida13) uma ontologia de aplicação que, caso seja utilizada por um DBA ou ferramenta, pode inferir ações de sintonia fina de forma automática ou semiautomática, especificamente para a criação, remoção ou reindexação de índices em bancos de dados relacionais. Esta ontologia de aplicação é dividida em duas outras ontologias: uma ontologia de domínio que descreve os conceitos envolvidos na tarefa de sintonia fina de índices (ex.: consulta, tabela, coluna) e uma ontologia de tarefas que contém heurísticas capazes de analisar a carga de trabalho instanciada na ontologia e inferir quais ações de sintonia fina poderiam ser realizadas nesse contexto.

O objetivo desta referida tese de doutorado não foi mudar as soluções propostas por heurísticas já existentes, mas sim, melhorar a eficácia da prática de sintonia fina. A contribuição inovadora da proposta foi a de acrescentar semântica ao processo de sintonia fina guiado por modelos definidos em uma ontologia e oferecer transparência e confiabilidade acerca das alternativas disponíveis para possíveis cenários no SGBD, por meio de justificativas concretas para as decisões que foram tomadas. Além disso, buscou-se viabilizar a geração automática de novas práticas de sintonia fina, que podem ser inferidas a partir das práticas existentes ou de novas regras e conceitos que venham a serem propostos. Esta abordagem permite também a realização de combinações de heurísticas de sintonia fina através da ontologia de aplicação.

A ontologia proposta (Almeida13) formaliza o domínio de sintonia fina em banco de dados relacionais e os conceitos envolvidos na criação e manutenção de índices, com a finalidade de possibilitar a justificativa da sintonia fina de banco de dados e flexibilizar, em alto nível, as alternativas a serem aplicadas no raciocínio de sintonia. A ontologia inicialmente, teve o foco sobre a estrutura de índices de banco de dados. Assim, foram contempladas na ontologia de tarefas duas heurísticas: uma para a criação de índices hipotéticos e outra para, dentre os índices hipotéticos

criados, sugerir quais índices reais devem ser criados baseados na carga de trabalho instanciada.

Também foi apresentada na mesma tese de doutorado (Almeida13) uma proposta de um *framework* chamado Outer-Tuning para executar a sua ontologia de aplicação. A abordagem básica do Outer-Tuning é poder capturar a carga de trabalho do banco de dados e, através de uma máquina de regras, instanciar os conceitos da ontologia e obter as ações de sintonia fina inferidas pelas regras.

Seu nome se dá por uma analogia com o outer-join, onde ao se fazer uma junção, as tuplas de uma tabela que não correspondem com as tuplas da outra tabela são apresentadas no resultado final. Analogamente, as alternativas de sintonia fina não utilizadas na solução final também são apresentadas, bem como as justificativas do seu não uso. Porém, devido ao escopo de sua tese, o *framework* Outer-Tuning não pôde ser desenvolvido. Os testes apresentados foram gerados através de simulações e exemplos abstratos e provas de conceito.

3.4 Conclusões

Além da dificuldade na seleção das melhores soluções de sintonia fina, o DBA enfrenta o desafio de justificar as decisões que são tomadas para melhorar o desempenho da execução de uma determinada carga de trabalho. Não há uma ferramenta integrada que auxilie com argumentos suficientes para justificar as soluções escolhidas pelas ferramentas de sintonia fina, sejam automáticas ou semiautomáticas.

Faz parte do trabalho de um DBA descrever todas ou, se inviável, a maioria das soluções disponíveis e avaliadas em uma atividade de sintonia fina. Um software que realize a sintonia fina de banco de dados, que apresente as alternativas que foram pensadas e que, ao mesmo tempo, fundamente a escolha de cada uma, trará uma maior confiança em relação ao uso de ferramentas automáticas e facilitará o trabalho de avaliação da tarefa de sintonia fina feita pelo DBA.

Apesar da quantidade significativa de pesquisas que abordam o tema de seleção de visões materializadas, nenhuma delas aborda a proposta dessa dissertação. Todas as pesquisas encontradas se preocupam em propor ou evoluir algoritmos para se aproximar de uma solução ótima para o problema. Porém nenhuma delas prevê em seu escopo, justificativas eficientes para que o DBA possa escolher, dentre as soluções encontradas, quais ele deve optar.

Este capítulo apresentou as principais pesquisas do grupo BioBD, as principais heurísticas encontradas para a seleção e criação de visões materializadas, e a pesquisa (Almeida13) que propõe uma ontologia para inferir ações de sintonia fina para bancos de dados.

4
Extensão da ontologia: visões materializadas

A ontologia de sintonia fina proposta por Ana Carolina Brito de Almeida (Almeida13) contemplou apenas ações de criação e manutenção de índices. Assim, mesmo previsto e proposto para o *framework* Outer-Tuning, não se permitia combinar estratégias de sintonia fina, ou seja, utilizar mais de um tipo de sintonia fina de forma simultânea e complementar, como por exemplo, criar visões materializadas e também índices sobre estas mesmas visões materializadas. Também não foi devidamente documentado na referida tese de doutorado um passo-a-passo de um possível processo de extensão da ontologia de aplicação.

Considerando estas características da pesquisa, aliado ao fato do laboratório BioBD já ter trabalhado com a seleção e criação de VMs 3, foi natural o surgimento da proposta de extensão da ontologia de sintonia fina para a seleção e criação de visões materializadas e a documentação do processo de extensão para que pudesse ser usado por outros pesquisadores que queiram incluir novas heurísticas de sintonia fina na ontologia.

Durante esta pesquisa, foi realizada e documentada a extensão da ontologia de aplicação. Primeiro foi realizada a extensão da ontologia de domínio para a inclusão dos conceitos envolvidos na tarefa de seleção e criação de VMs. A partir daí, foram adicionadas na ontologia de tarefas três heurísticas de sintonia fina: uma para a criação de visões materializadas hipotéticas (VMH), e duas outras para decidir dentre todas as VMHs possíveis, quais VMs deveriam ser persistidas no banco de dados. Estas extensões serão discutidas em detalhes neste capítulo.

Para fins de comparação, apresentamos na Figura 4.1 a ontologia original proposta por (Almeida13) e em seguida, a Figura 4.2 a ontologia estendida com todas as modificações feitas durante a pesquisa apresentada nessa dissertação de mestrado. Estas extensões serão discutidas em detalhes neste capítulo.

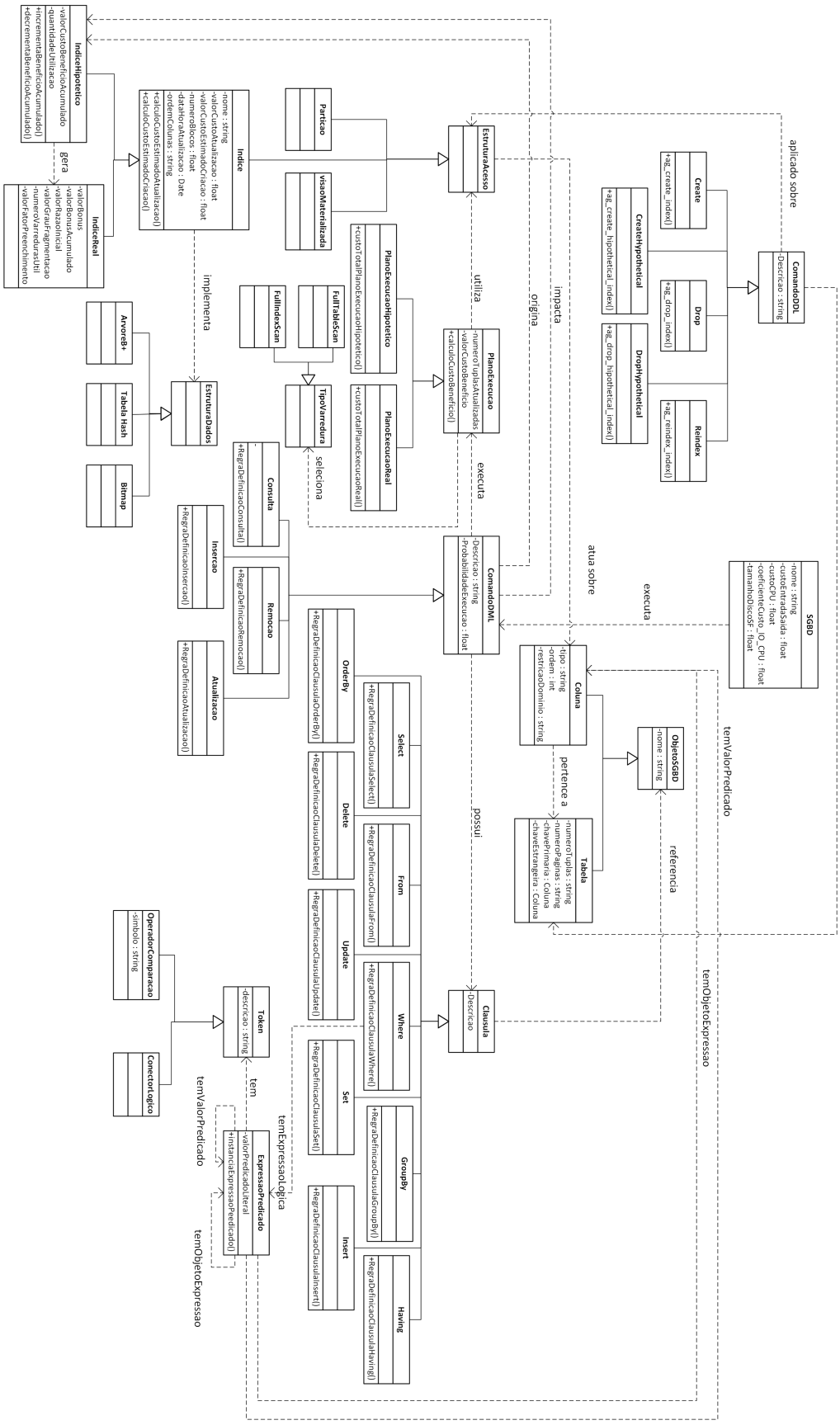

Figura 4.1: Ontologia Original proposta por (Almeida2013)

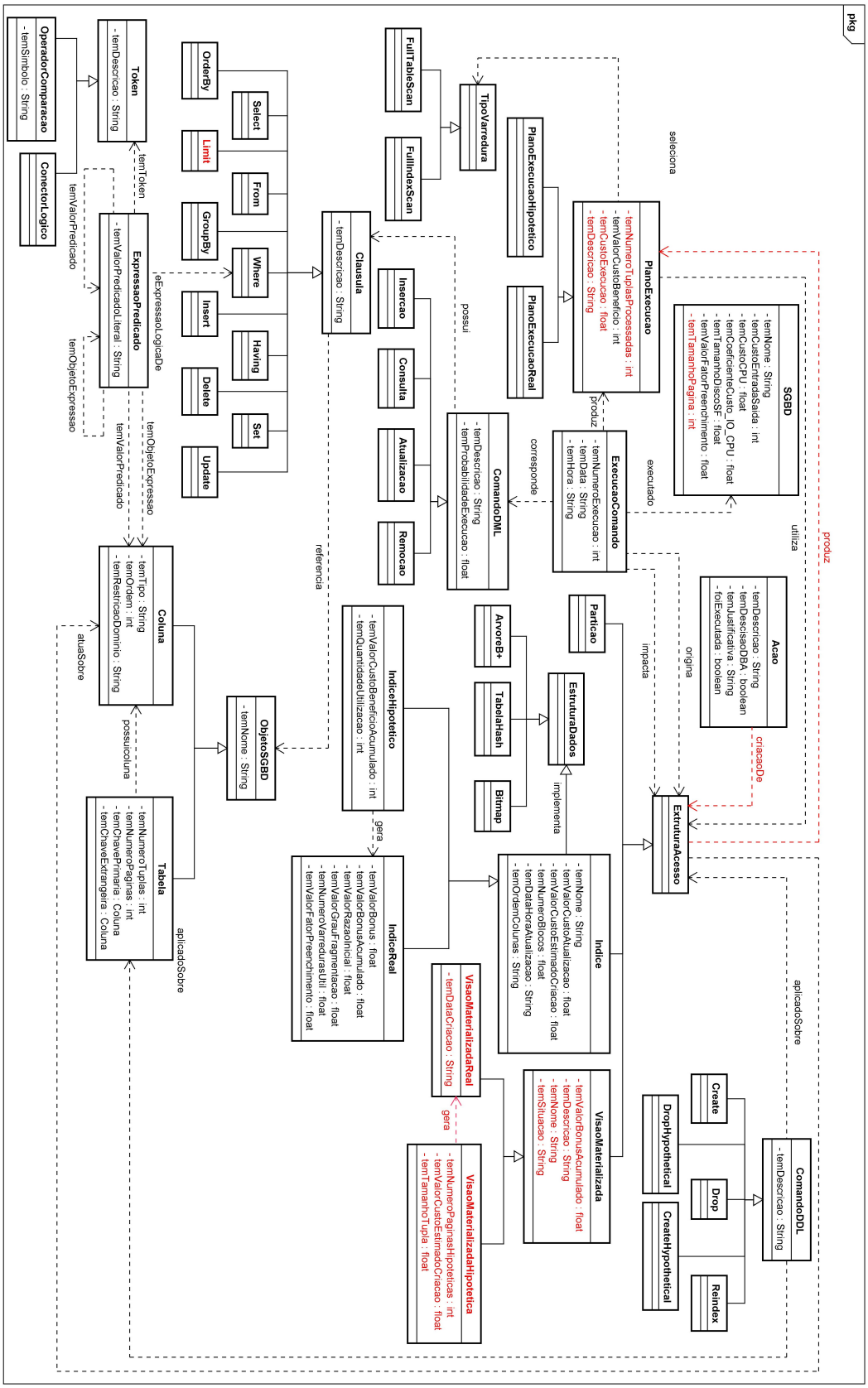

Figura 4.2: Ontologia de domínio estendida para seleção e criação de visões materializadas

4.1 Extensão da ontologia de domínio

Para realizar a extensão da ontologia de aplicação proposta (Almeida13), foram adicionadas à ontologia de domínio novos conceitos, atributos e relações. A Figura 4.2 exibe a ontologia de domínio estendida para realizar a seleção e criação de visões materializadas.

Para representar as estruturas de acesso utilizadas no processo de seleção de visões materializadas, foram criados os conceitos para representar uma VMH (*VisaoMaterializadaHipotetica*) e uma visão materializada real (VMR) (*VisaoMaterializadaReal*) na ontologia de domínio (Figura 4.2). Ambos são especializações do conceito *VisaoMaterializada*. Suas inclusões foram necessárias para que a ontologia de tarefas pudesse manipular indivíduos desses conceitos durante a execução das novas heurísticas de sintonia fina.

Foram criados na ontologia de domínio os conceitos *VisaoMaterializada*, *VisaoMaterializadaReal* e *VisaoMaterializadaHipotetica* (Figura 4.3) para instanciarem os conceitos necessários à tarefa de seleção de visões materializadas. Para que o conceito *VisaoMaterializada* pudesse ser utilizado durante a extensão da ontologia de domínio, foi necessária a inclusão dos quatro atributos descritos na Tabela 4.1. A inclusão é justificada pelo uso da herança entre os conceitos *temValorBonusAcumulado*, *temDescricao*, *temNome*, *temSituacao*, já que tanto uma VMH quanto uma VMR possuem estas mesmas informações.

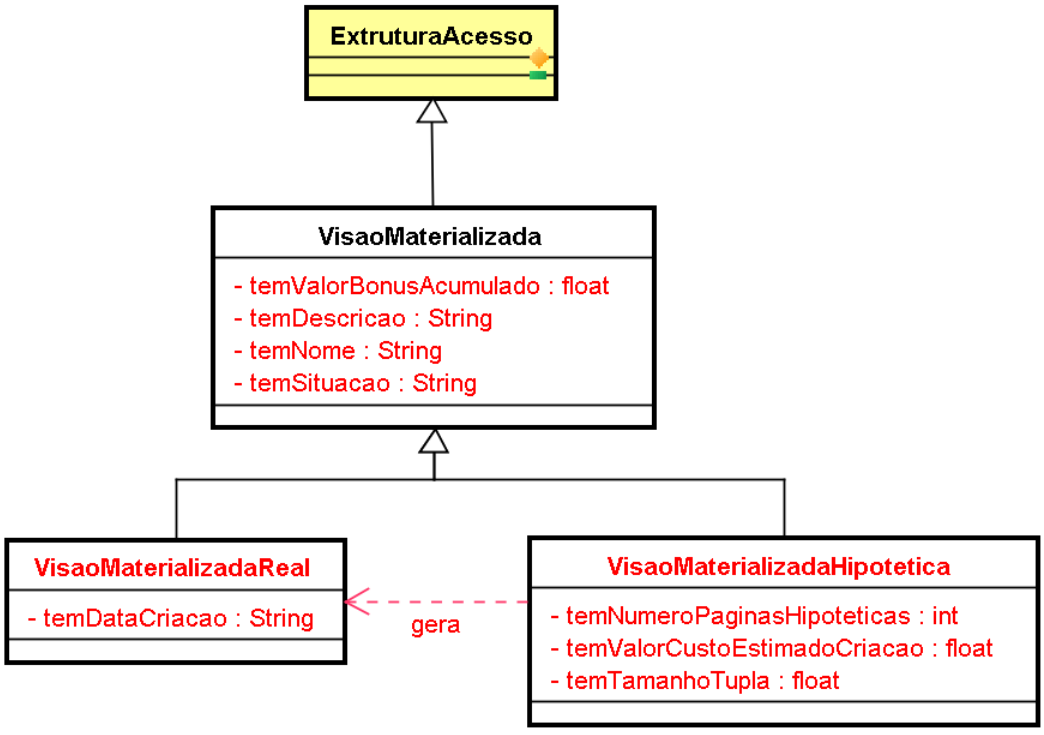

Figura 4.3: Novos conceitos e atributos adicionados à Ontologia de Domínio

temValorBonusAcumulado	Acumula o bônus (positivo ou negativo) de uma VM. Ele é incrementado ou decrementado por regras das heurísticas de criação de VMs.
temDescricao	Armazena a definição da VM.
temNome	Armazena o nome da VM que foi/será criada no banco de dados.
temSituacao	Mostra o estado da VM em relação ao banco. Possui três valores possíveis: "*criada pelo DBA*", "*descartada pelo DBA*" e "*aguardando decisão do DBA*".

Tabela 4.1: Atributos adicionados ao conceito VisaoMaterializada

Como mostrado na Figura 4.2, também foi adicionado o relacionamento *gera* entre os conceitos *VisaoMaterializadaHipotetica* e *VisaoMaterializadaReal* que como os nomes sugerem, determina que uma *VisaoMaterializadaReal* é gerada a partir de uma *VisaoMaterializadaHipotetica*.

Além dos conceitos descritos anteriormente, também foram adicionados os seguintes atributos aos conceitos já existentes na ontologia de domínio exibidos na Tabela 4.2.

Conceitos existentes	Atributos adicionados	Descrição
PlanoExecucao	**temNumeroTuplas Processadas**	Armazena o número de tuplas processadas por um comando SQL. É extraído do plano de execução fornecido pelo banco de dados para cada execução de um comandoDML.
PlanoExecucao	**temDescricao**	Armazena no formato de texto, o plano de execução completo fornecido pelo SGBD.
PlanoExecucao	**temCustoExecucao**	Armazena o custo de execução de um comando DML, obtido pelo plano de execução fornecido pelo SGBD.
SGBD	**temTamanhoPagina**	Armazena o tamanho da página que o SGBD aplica no armazenamento.

Tabela 4.2: Atributos adicionados aos conceitos já existentes na ontologia de domínio

O atributo *temCustoExecucao* (Tabela 4.2) do conceito *PlanoExecucao* é um novo nome dado ao atributo *temValorCustoBeneficio*. Ele foi renomeado da versão original da ontologia para que o nome refletisse melhor o dado que armazena.

O atributo *temDescricao* (Tabela 4.2) do conceito *PlanoExecucao* armazena o plano de execução de um comando SQL gerado pelo SGBD. Valores extraídos do plano de execução são armazenados em atributos próprios como o custo de execução (*temCustoExecucao*) e o número de tuplas processadas (*temNumeroTuplasProcessadas*). Porém, decidiu-se por armazenar o plano de execução para que futuras heurísticas pudessem utilizar outras informações contidas nele sem a necessidade de uma nova extensão da ontologia de domínio. Por exemplo, uma heurística que avalia os tipos de junções de uma consulta (*hash-join*, *merge-join*, *nested-loop join*, entre outros) podem extrair essa informação do plano de execução armazenado no atributo *temDescricao* do conceito *PlanoExecucao*.

O atributo *temTamanhoPagina* do conceito SGBD, foi adicionado para ser usado no cálculo que prevê o tamanho de uma visão materializada hipotética caso venha a ser persistida. Ele é obtido através de uma regra SWRL e explicado na Seção 4.2.

Por fim, foi adicionado o relacionamento produz entre os conceitos *EstruturaAcesso* e *PlanoExecucao*. A ideia é formalizar o relacionamento entre os planos de execução e as estruturas de acesso utilizadas. Por exemplo, são os planos de execução hipotéticos (*PlanoExecucaoHipotetico*) gerados para cada visão materializada hipotética (*VisaoMaterializadaHipotetica*) pela heurística de visões materializadas hipotéticas (Seção 4.2).

4.2 Extensão da ontologia de tarefas

A ontologia de tarefas foi estendida com a inserção de três novas heurísticas: (i) a heurística de visões materializadas hipotéticas (HVMH), que é responsável por criar uma VMH para cada consulta capturada na carga de trabalho; (ii) a heurísticas de benefícios (HB) responsável por escolher, dentre todas as visões materializadas hipotéticas, quais deveriam se tornar visões materializadas reais; e (iii) a heurística de expectativas (HE), que tem a mesma função da heurística de benefícios, mas utiliza uma estratégia diferente.

4.2.1 Inclusão da heurística de visões materializadas hipotéticas (HVMH)

A heurística de visões materializadas hipotéticas foi proposta em (Carvalho11) descrita na Seção 3. Dada uma carga de trabalho qualquer, a HVMH cria uma visão materializada hipotética para cada consulta SQL da carga de

trabalho. A partir daí, cada vez que uma visão materializada hipotética é considera apta para uso no processamento de uma consulta, a VMH acumula um benefício (caso melhore o desempenho da consulta) ou um malefício (caso contrário) como explicado na seção 2.11

A escolha por essa heurística se deu pela similaridade da estratégia de criação dos índices hipotéticos que já estava implementada na ontologia pela pesquisa de (Almeida13). Essa característica foi utilizada para facilitar a coexistência da seleção das duas estruturas de acesso (índices e VMs) simultaneamente pela ontologia, assim como uma futura combinação das duas abordagens.

A HVMC adaptada e inserida na ontologia possui três passos básicos: (i) instanciação da carga de trabalho; (ii) geração do comando de criação da VM; e (iii) instanciação da VMH.

Instanciação da carga de trabalho

A Figura 4.4 mostra um relacionamento entre os conceitos *Heuristica* e *Conceito* chamado *PreCondicao*. Esse relacionamento formaliza a necessidade de heurísticas possuírem conceitos previamente instanciados para a sua execução. Por exemplo: para que a HVMH crie visões materializadas hipotéticas é necessário que haja, entre outros, indivíduos do tipo *Consulta* instanciados na ontologia. Essa necessidade é formalizada através do relacionamento *PreCondicao* da ontologia de tarefas (Figura 4.4).

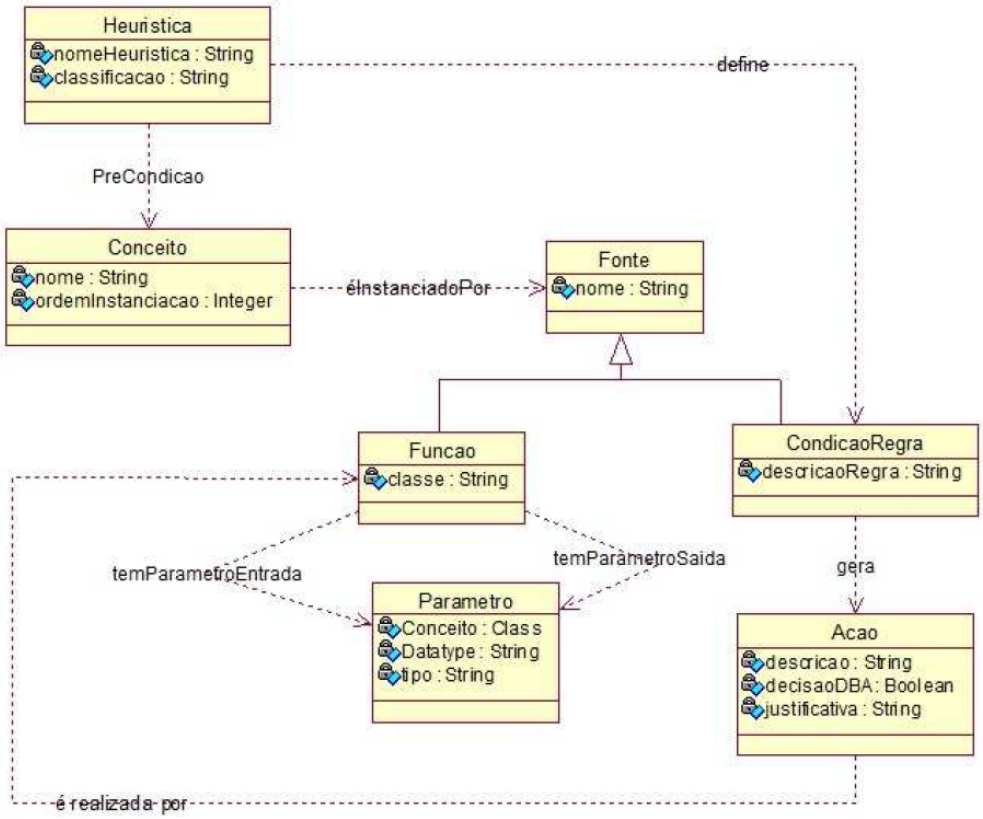

Figura 4.4: Ontologia de tarefa (Almeida13)

Cada heurística possui um conjunto de conceitos que são pré-condições para sua execução. Eles são extraídos da carga de trabalho instanciada na ontologia de domínio. Uma das formas de se descobrir quais conceitos devem ser instanciados, de acordo com as heurísticas contidas na ontologia de tarefas, é realizar uma consulta à máquina de regras que instanciou a ontologia através da regra SWRL contida na Figura 4.5.

```
1  Heuristica(?heu)   ∧
2          temPreCondicao(?heu, ?con)   ∧
3          temNome(?con, ?nome)   ∧
4          temOrdemInstanciacao(?con, ?ordem)   ∧
5          eInstanciadoPor(?con, ?funcao)   ∧
6          temNome(?funcao, ?nomeFuncao)   ∧
7          temClasse(?funcao, ?classe)   ∧
8          temBiblioteca(?funcao, ?biblioteca)   →
9   sqwrl:select(?heu, ?con, ?nome, ?nomeFuncao, ?classe, ?ordem,
   ?biblioteca)   ∧   sqwrl:orderBy(?ordem)
```

Figura 4.5: Regra para captura de conceitos pré-condições das heurísticas

A regra mostrada na Figura 4.5 seleciona todos os conceitos que são pré-condição de uma heurística (linhas 1 e 2), o nome do conceito (linha 3) e a sua ordem de instanciação (linha 4).

A ordem de instanciação é usada em casos em que conceitos são dependentes de outros conceitos. Por exemplo: A HVMH possui a pré-condição que o conceito *Coluna* e *Tabela* devam estar instanciados na ontologia para que a heurística seja executada. Através da regra exibida na Figura 4.5, poderá se ter a informação que é necessário que indivíduos do tipo *Tabela* e *Coluna* estejam instanciado na ontologia, e que há uma ordem de instanciação que oriente o indivíduo *Tabela* ser instanciado antes do conceito *Coluna*. A regra também traz os dados relativos à forma que deve ser feita a extração da carga de trabalho (linhas 5, 6, 7 e 8). Essa forma de extração é detalhada no Capítulo 5 que fala sobre o *framework* Outer-Tuning e o método de execução da ontologia.

A regra SWRL exibida na Figura 4.5 retorna os conceitos que são pré-condição das heurísticas de sintonia fina. Através dessa consulta é possível obter quais conceitos, atributos e relacionamentos são necessários para serem (i) extraídos da carga de trabalho e (ii) instanciados na ontologia de domínio para a execução das heurísticas e geração das ações de sintonia fina.

Geração do comando de criação da visão materializada

Para cada consulta capturada na carga de trabalho analisada, a HVMH instancia um indivíduo do tipo visão materializada hipotética (*VisaoMaterializadaHipotetica*). Para criar este indivíduo, a consulta capturada é reescrita através do Algoritmo 1 e transformada em um comando DDL para a criação da visão materializada no banco de dados. Vale ressaltar que a HVMH cria uma visão materializada hipotética, portanto, nada é criado fisicamente no banco de dados pela HVMH. Uma VMH só se torna real (persistida no banco de dados) quando uma outra heurística decide gerar uma visão materializada real a partir da VMH analisada.

Como dito na Seção 4.1, o conceito *VisaoMaterializadaHipotetica* foi um dos conceitos incorporados à ontologia de domínio (Figura 4.2). Ele possui um atributo chamado *temDescricao* que armazena a definição do comando SQL de criação da visão materializada. Esse comando de criação da VM é definido através de um algoritmo baseado no algoritmo *DefineView* (Carvalho11). Foram feitas adaptações para a sua execução no contexto da ontologia mas sua função ainda é a de reescrever a consulta SQL capturada no banco de dados para torna-la genérica o suficiente para que outras consultas semanticamente similares possam usar uma mesma VM.

Dado uma consulta capturada no banco de dados, a HVMH prevê os seguintes passos para reescrita da consulta SQL para a criação da definição de uma VM:

Entrada: Uma consulta SQL
Saída: Um comando DDL para criação de uma visão materializada
inicio
I. Adicionar o comando de criação da VMH;
II. Tornar os atributos de projeção da consulta em atributos de projeção da VMH;
III. Tornar todas as Tabelas referenciadas na consulta em Tabelas referenciadas na VMH;
IV. Tornar atributos de filtro da consulta em atributos de projeção da VMH;
V. Se houver, tornar atributos de having da consulta em atributos de projeção da VMH;
VI. Se houver, definir *group by* através dos atributos de projeção não agregados da VMH;
VII. Se houver, tornar as restrições de chaves estrangeira da consulta em restrições de chaves estrangeira na VMH;
fin

Algoritmo 1: Algoritmo DefineView da heurística HMVH
.

O principal objetivo desse algoritmo é tornar a definição da VMH genérica o suficiente para outras consultas que utilizem os mesmos atributos. Caso existam outras consultas semelhantes à que originou a VMH mas, com restrições diferentes, elas poderiam usar a mesma VMH porém com as suas próprias restrições. Isso faz com que, uma única visão materializada possa responder a uma maior quantidade de consultas da carga de trabalho e, consequentemente, tenha um benefício acumulado maior e um menor custo de manutenção, já que diversas consultas podem ser respondidas ao custo de criação e manutenção de uma única visão.

Exemplo de execução da HVMH

Como exemplo de execução da tarefa de reescrita de consulta da HVMH, considere a seguinte consulta SQL como entrada:

```
1 SELECT nome_vendedor, sum(valor)
2 FROM vendas, vendedor
3 WHERE
4     vendas.id_vendedor = vendedor.id_vendedor
5    AND uf_venda = 'GO'
6 HAVING count(id_produto) > 5
7 GROUP BY nome_vendedor;
```

O passo I será adicionar o comando de criação da VMH:

```
1 CREATE MATERIALIZED VIEW V_EXEMPLO_HEURISTICA AS
```

O passo II será tonar os atributos de projeção da consulta em atributos de projeção da VMH:

```
1 CREATE MATERIALIZED VIEW V_EXEMPLO_HEURISTICA AS
2
3 SELECT nome_vendedor, sum(valor)
```

Já o passo III, será tornar todas as tabelas referenciadas na consulta em tabelas referenciadas na VMH, no exemplo a tabela vendas e vendedor:

```
1 CREATE MATERIALIZED VIEW V_EXEMPLO_HEURISTICA AS
2
3 SELECT nome_vendedor, sum(valor)
4
5 FROM vendas, vendedor
```

No passo IV, tornam-se atributos de filtro da consulta em atributos de projeção da VMH:

```
1 CREATE MATERIALIZED VIEW V_EXEMPLO_HEURISTICA AS
2
3 SELECT nome_vendedor, sum(valor), uf_venda
4
5 FROM vendas, vendedor
```

Seguindo o passo V, tornam-se os atributos da cláusula *having* da consulta em atributos de projeção da VMH:

```
1 CREATE MATERIALIZED VIEW V_EXEMPLO_HEURISTICA AS
2
3 SELECT nome_vendedor, sum(valor), uf_venda, id_produto
4
5 FROM vendas, vendedor
```

No passo VI define-se o *group by* através dos atributos de projeção não agregados da VMH:

```
1  CREATE MATERIALIZED VIEW V_EXEMPLO_HEURISTICA AS
2
3  SELECT nome_vendedor, sum(valor), uf_venda, id_produto
4
5  FROM vendas, vendedor
6
7  GROUP BY nome_vendedor, uf_venda, id_produto
```

E por último, o passo VII adiciona uma cláusula *where* com as restrições de chave estrangeira:

```
1  CREATE MATERIALIZED VIEW V_EXEMPLO_HEURISTICA AS
2
3  SELECT nome_vendedor, sum(valor), uf_venda, id_produto
4
5  FROM vendas, vendedor
6
7  WHERE vendas.id_vendedor = vendedor.id_vendedor
8
9  GROUP BY nome_vendedor, uf_venda, id_produto
```

Após todos os passos da HVMH, tem-se então a definição completa da visão materializada segundo a heurística.

Restrições da linguagem SWRL para executar o algoritmo *DefineView*

A linguagem de regras SWRL (escolhida para linguagem de consultas na ontologia original) não possui os operadores lógicos *or* e *not*. Isso trouxe uma dificuldade para a implementação do algoritmo *DefineView* através de regras SWRL. A ausência dos operadores *or* e *not* aliado à característica de que uma mesma consulta pode ser escrita de diversas formas, obriga a escrita de uma regra SWRL para cada possível formação sintática da consulta.

Para ilustrar o problema, podemos considerar as cláusulas SQL *select*, *from*, *where*, *group by*, *order by*, *having*, *fetch first*, compatíveis com a versão SQL:2011 (ISO2015). Dentre estas, as cláusulas obrigatórias de uma consulta são a *select* e from. Logo, das 7 cláusulas 5 podem ou não ocorrer. Considerando que o comando SQL sempre venha com as cláusulas na mesma ordem e considerando apenas as cláusulas *select*, *from*, *where*, *group by*, *order by, having, fetch first*, seriam necessárias 32 regras SWRL para tratar cada possível arranjo da consulta SQL, já que não há a possibilidade de em uma mesma regra SWRL testar a presença ou não de uma cláusula (usando o operador *not*).

Diante disso, utilizou-se do conceito *Funcao* da ontologia de tarefas para execução do algoritmo *DefineView* 1. Esse conceito *Funcao* foi criado na ontologia de tarefas para permitir que operações que tenham complexidade impeditivas de serem expressas por regras, sejam realizadas fora da máquina de regras e tenham seus resultados instanciados. Isso permite que algoritmos como o *DefineView*, ou ainda mais complexos, sejam implementados e utilizados pela ontologia para o processo de inferência. A forma que foi implementado este recurso está descrito no Capítulo 5.

O conceito *Funcao* da ontologia de tarefas já tinha sido previsto na versão original da ontologia (Almeida13) para executar procedimentos inviáveis de ser expressos por regras. A decisão de utilizar o conceito *Funcao* e as bibliotecas de funções para executar o algoritmo *DefineView* de reescrita da consulta foi a forma encontrada para superar uma dificuldade no uso da linguagem SWRL, a falta de operadores *not* e *or*, e evitar o trabalho braçal de escrever uma regra SWRL para cada combinação possível de cláusulas de um comando SQL. Porem isso trouxe como prejuízo a dependência de uma biblioteca de código fonte e consequentemente uma diminuição no controle da ontologia sobre os passos executados para a inferência das ações de sintonia fina.

4.2.2 Inclusão da heurística de benefícios para seleção de visões materializadas

A segunda heurística inserida na ontologia de tarefas foi a heurística de benefícios (HB) usada em (Salles04), (Morelli06), (Monteiro08) e (Carvalho11) para definir quais das visões materializadas hipotéticas poderiam trazer benefícios para a carga de trabalho, caso sejam efetivamente persistidas no banco de dados.

O conceito VMH possui o nome *VisaoMaterializadaHipotetica* na ontologia e foi um dos conceitos adicionados durante a extensão da ontologia de domínio assim como *VisaoMaterializadaReal*. Ambos são especificações do conceito *VisaoMaterializada* (Figura 4.3).

Para determinar o momento de instanciação de um conceito *VisaoMaterializadaReal*, uma instância do conceito *VisaoMaterializadaHipotetica* (criada pela heurística HVMH) correspondente mantém um somatório do benefício associado a uma *VisaoMaterializadaHipotetica* que é o ganho que a VMH traria caso fosse uma visão materializada real. O benefício é calculado através da diferença entre o custo de execução da consulta que originou a *VisaoMaterializadaHipotetica* e o custo de execução estimado ao executar a mesma consulta agora na presença da *VisaoMaterializadaHipotetica*.

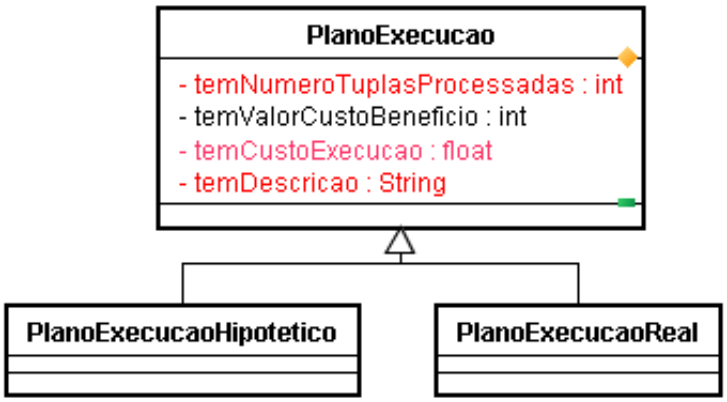

Figura 4.6: Conceito *PlanoExecucao* e suas especializações *PlanoExecucaoHipotetico* e *PlanoExecucaoReal*

Para se obter o custo de execução hipotético, cria-se através da regra SWRL exibida na Figura 4.6, uma instância do conceito *PlanoExecucaoHipotetico* (Figura 4.6).

O conceito *PlanoExecucaoHipotetico* já existia na ontologia. Porém, foram adicionadas os atributos *temNumeroTuplasprocessadas*, *temCustoExecucao*, *temDescricao* e descritos na Seção 4.2.

A instanciação do conceito *PlanoExecucaoHipotetico* é realizado pela heurística de benefícios. Para a instanciação, é utilizada a regra SWRL exibida na Figura 4.7.

A Figura 4.7 exibe a regra SWRL responsável por instanciar um plano de execução. Nas linhas de 1 a 14 são verificadas as condições de existência dos indivíduos necessários para a execução do cálculo do custo de execução estimado. As linhas 15 a 21 realizam o cálculo do custo hipotéticos de execução e do benefício hipotético calculado pelo plano de execução para a execução do comando na presença da visão materializada hipotética. Por fim, as linhas 22 a 25 instanciam um indivíduo do conceito *PlanoExecucaoHipotetico* com os seus atributos.

A heurística de benefícios instancia um plano de execução hipotético para cada consulta capturada e que pode ser respondida utilizando uma VMH. Após o plano de execução hipotético ser instanciado, a heurística de benefícios também atualiza as informações da VMH instanciada.

Um dos atributos calculados e instanciados pela regra SWRL da Figura 4.7 é o valor do custo hipotético de execução (linha 23) de uma consulta, na presença de uma visão materializada hipotética calculado como tendo o custo de uma varredura simples (*fullscan*) em uma VM (Carvalho11). O custo de uma varredura simples pode ser calculado como a quantidade de páginas da tabela ou visão materializada (Hose09). Para calcular a quantidade de páginas hipotéticas, utilizou-se a fórmula:

```
1  ExecucaoComando(?exec) ∧
2     produz(?exec, ?planoR) ∧
3     executado(?exec, ?bd) ∧
4     origina(?exec, ?vmH) ∧
5  PlanoExecucaoReal(?planoR) ∧
6     temCustoExecucao(?planoR, ?custoReal) ∧
7  SGBD(?bd) ∧
8     temValorFatorPreenchimento(?bd, ?fator) ∧
9     temTamanhoPagina(?bd, ?tamPag) ∧
10 VisaoMaterializadaHipotetica(?vmH) ∧
11    produz(?vmH, ?plan) ∧
12    temTamanhoTupla(?vmH, ?tamTuplas) ∧
13 PlanoExecucao(?plan) ∧
14    temNumeroTuplasProcessadas(?plan, ?numTuplas) ∧
15    swrlb:multiply(?num1, ?numTuplas, ?tamTuplas) ∧
16    swrlb:multiply(?num2, ?num1, ?fator) ∧
17    swrlb:divide(?custoH, ?num2, ?tamPag) ∧
18    swrlb:ceiling(?custoHTemp, ?custoH) ∧
19    swrlb:multiply(?custoHtotal, 2, ?custoHTemp) ∧
20    swrlb:subtract(?ganho, ?custoReal, ?custoHtotal) ∧
21    swrlx:makeOWLIndividual(?planH, ?ganho) →
22 PlanoExecucaoHipotetico(?planH) ∧
23    temCustoExecucao(?planH, ?custoHtotal) ∧
24    produz(?exec, ?planH) ∧
25    temValorCustoBeneficio(?planH, ?ganho)
```

Figura 4.7: Regra SWRL para criação de indivíduos do conceito PlanoExecucaoHipotetico

$$QtdPagHipo = \frac{numHipoTuplas * tamHipoTupla * fatorPreenchimento}{tamPagSGBD} \tag{4-1}$$

.

A fórmula é inspirada no modelo de custos usado pelos SGBDs para o cálculo do tamanho em páginas do resultado de uma consulta durante a criação do plano de execução (Hose09). Para o cálculo do tamanho hipotético de uma visão materializada, dada a sua definição e o plano de execução da consulta de sua definição, seus termos são preenchidos para o cálculo da seguinte forma:

- **numHipoTuplas**:O número de tuplas estimado pelo plano de execução que a visão materializada teria se vier a ser persistida no banco de dados.
- **tamHipoTupla**:Cálculo do tamanho máximo que uma tupla da VM terá se todas as colunas vierem a ter seus valores completamente preenchidos.
- **fatorPreenchimento**: Fator de preenchimento da página do disco utilizado pelo SGBD. Tratado como *fillfactor* na documentação dos SGBDs, pode ser alterado pelo DBA nas configurações do SGBD. Serve para evitar que toda

a página de dados seja preenchida durante a sua alocação. Configuração útil para determinadas cargas de trabalho com muitos comandos de atualização dos dados.

- **tamPagSGBD**: Tamanho da página de dados utilizado pelo SGBD.

A Fórmula 4-1 foi representada por uma regra SWRL para calcular o número de páginas hipotéticas do conceito *VisaoMaterializadaHipotetica*, mostrada na Figura 4.8:

```
1  SGBD(?sgbd) ∧
2     temTamanhoPagina(?sgbd, ?tamPag) ∧
3     temValorFatorPreenchimento(?sgbd, ?fator) ∧
4  VisaoMaterializadaHipotetica(?vmH) ∧
5     temTamanhoTupla(?vmH, ?tamTuplas) ∧
6     produz(?vmH, ?plano) ∧
7  PlanoExecucaoReal(?plano) ∧
8     temNumeroTuplasProcessadas(?plano, ?numTuplas) ∧
9     swrlb:multiply(?num1, ?numTuplas, ?tamTuplas) ∧
10    swrlb:multiply(?num2, ?num1, ?fator) ∧
11    swrlb:divide(?num3, ?num2, ?tamPag) ∧
12    swrlb:ceiling(?total, ?num3) →
13 temNumeroPaginasHipoteticas(?vmH, ?total)
```

Figura 4.8: Regra SWRL para cálculo do custo estimado de números de páginas de uma VMH

Da linha 1 até a linha 8 são verificadas a existência de instâncias dos indivíduos e atributos necessários para o cálculo do custo estimado de números de página. As linhas 9 até 12 executam a Fórmula 4-1 e a linha 13 instancia o atributo de número de páginas hipotéticas para o indivíduo VMH instanciado.

Por meio dessa regra, a ontologia de tarefas consegue calcular o número de páginas hipotéticas para cada VMH criada e, consequentemente, calcular o bônus acumulado a cada execução da consulta que originou a VMH.

Após calcular o bônus acumulado para cada VMH, é necessário então estimar o custo de criação da visão materializada. Este é obtido como sendo o custo de uma varredura simples da visão materializada mais o custo de gravação das páginas em memória secundária, estimada como sendo equivalente a 2 (duas) vezes a quantidade de páginas hipotéticas da VMH (Monteiro08)(Morelli06)(Carvalho11)(Salles04).Esse cálculo é realizado pela regra exibida na Figura 4.9.

```
1  VisaoMaterializadaHipotetica(?vmH) ∧
2     produz(?vmH, ?plan) ∧
3  PlanoExecucaoReal(?plan) ∧
4     temCustoExecucao(?plan, ?valorConsultar) ∧
5     temNumeroPaginasHipoteticas(?vmH, ?pagHipo) ∧
6     swrlb:multiply(?pagHipoMult, ?pagHipo, 2) ∧
7     swrlb:add(?total, ?pagHipoMult, ?valorConsultar) ∧
8     swrlb:greaterThan(?total, 0) →
9  temValorCustoEstimadoCriacao(?vmH, ?total)
```

Figura 4.9: Regra SWRL para cálculo do custo estimado de criação de uma VMH

4.2.3 Instanciação de visões materializadas reais

A heurística de benefícios determina que uma estrutura de dados deve ser criada quando o seu bônus hipotético acumulado for maior que o seu custo estimado de criação. Portanto, a instanciação de uma *VisaoMaterializadaReal* acontece quando o bônus acumulado de uma VMH for maior que o seu custo de criação. Cada VMH possui um valor de bônus acumulado calculado através da Figura 4.9, e um valor de custo de criação definido pela Figura 4.10. Com os valores instanciados, a heurística de benefícios monitora esses valores e instancia uma *VisaoMaterializadaReal* toda vez que o bônus acumulado ultrapassa o custo de criação. Esse monitoramento é realizado através da Regra SWRL exibida na Figura 4.10.

```
1  VisaoMaterializadaHipotetica(?vmh) ∧
2        temSituacao(?vmh, ?situacaoH) ∧
3        swrlb:equal(?situacaoH, "coletando") ∧
4        temNome(?vmh, ?nomeVmh) ∧
5        temValorBonusAcumulado(?vmh, ?bonus) ∧
6        temValorCustoEstimadoCriacao(?vmh, ?criacao) ∧
7        temDescricao(?vmh, ?descricao) ∧
8        swrlb:greaterThan(?bonus, ?criacao) ∧
9        swrlx:createOWLThing(?vmr, ?vmh) →
10 VisaoMaterializadaReal(?vmr) ∧
11       eGeradoPor(?vmh, ?vmr) ∧
12       temNome(?vmr, ?nomeVmh) ∧
13       temValorBonusAcumulado(?vmr, 0) ∧
14       temDescricao(?vmr, ?descricao) ∧
15       temSituacao(?vmr, "criada")
```

Figura 4.10: Regra SWRL para cálculo do custo estimado de criação de uma VM

Das linhas 1 até 7 são validados os atributos necessários para a comparação entre o bônus acumulado e o custo de criação. A linha 8 realiza a comparação destes dois valores através da função *greaterThan* da biblioteca SWRL swrlb. As linhas 10 até 15 realizam a instanciação do indivíduo *VisaoMaterializadaReal*. Dessa forma, a heurística de benefícios implementada através das regras criará uma instância do conceito *VisaoMaterializadaReal* assim que o bônus acumulado for maior que o custo de criação da visão materializada.

4.2.4 Heurística de expectativas: uma nova abordagem para seleção de visões materializadas

O *framework* Outer-Tuning foi planejado para executar concorrentemente mais de uma heurística para o mesmo tipo de sintonia fina. Essa característica permite a comparação entre as diferentes estratégias e a aderência de cada uma à carga de trabalho. O DBA pode avaliar, comparar, adequar e combinar as heurísticas para obter um melhor desempenho e/ou vazão para a carga de trabalho.

Para experimentar o uso de mais de uma heurística para o mesmo tipo de sintonia fina, é proposta nesta dissertação uma nova heurística para a criação de visões materializadas. Existem outras heurísticas de criação de VMs interessantes na literatura (vide seção 2.10, mas decidiu-se por criar uma nova. Isso porque

deseja-se avaliar aqui, através do Outer-Tuning, uma heurística desconhecida e que não houvesse histórico e nem resultados esperados. Isso ajudaria a aproximar os testes de uma situação mais realista, onde o DBA realiza tarefas de sintonia fina sem saber como as heurísticas se comportarão com a sua carga de trabalho.

Lembramos que não tem por objetivo encontrar melhores resultados de sintonia fina com a adição de uma nova heurística à ontologia de sintonia fina mas, sim, exercitar a capacidade de avaliação do Outer-Tuning frente a uma nova estratégia de sintonia fina, seja ela boa ou ruim.

A heurística proposta aqui foi denominada Heurística de Expectativas (Hexp). Esta denominação se justifica pela hipótese utilizada de não considerar a frequência dos comandos SQL na carga de trabalho (como faz a heurística de benefícios) mas apenas a expectativa de ganhos, dados os custos estimados de execução e criação de uma visão materializada hipotética.

Para a construção da nova heurística, partiu-se da seguinte hipótese do que seria uma boa consulta para justificar a criação de uma visão materializada: **se o custo de ler o resultado armazenado em memória secundária for 50% menor que o custo total de executar a consulta original, uma VM pode trazer economia de tempo para executar a carga de trabalho**. É uma premissa simplória, mas válida. Se o maior custo de uma consulta for o custo de CPU, ou a procura na memória secundária pelos dados necessários, com a materialização do resultado este custo seria ignorado e poderia haver uma economia utilizando-se uma VM.

Para descobrir qual a porcentagem do custo de execução da consulta é composta por CPU, a heurística extrai o (i) custo de execução total e (ii) o custo de leitura de páginas de disco do plano de execução da consulta avaliada. Através destes dados, calcula-se qual a porcentagem do custo total de execução é relativo à leitura de páginas de disco. Logo, se o custo de leituras de páginas em disco da visão materializada hipotética for 50% (ou qualquer outro valor definido como parâmetro pelo DBA) menor que a execução da consulta original, a heurística determina que aquela visão materializada seja efetivamente criada no banco de dados.

Com a regra definida, realizou-se a extensão da ontologia para a nova heurística de criação de visões materializadas. Como a heurística utiliza conceitos já utilizados pela heurística de benefícios, não foi necessário nenhum outro conceito novo na ontologia de domínio e nem nas regras de instanciação. Com isso, o processo de extensão se resumiu à instanciação de um novo indivíduo do tipo Heuristica na ontologia de tarefas, a inclusão dos conceitos *VisaoMaterializadaHipotetica* e *PlanoExecucao* como uma pré-condições para a execução da nova heurística, e a implementação da regra SWRL que reproduza a regra descrita na Figura 4.11.

```
1  SGBD(?bd) ∧
2        temTamanhoDiscoSF(?bd, ?tamSF) ∧
3  ComandoDML(?dml) ∧
4        corresponde(?exec, ?dml) ∧
5  ExecucaoComando(?exec) ∧
6        origina(?exec, ?vmh) ∧
7        produz(?exec, ?planoReal) ∧
8  PlanoExecucaoReal(?planoReal) ∧
9        temCustoExecucao(?planoReal, ?custoReal) ∧
10 VisaoMaterializadaHipotetica(?vmh) ∧
11       temNome(?vmh, ?nomeVmh) ∧
12       temDescricao(?vmh, ?descricao) ∧
13       temNumeroPaginasHipoteticas(?vmh, ?paginas) ∧
14       temValorCustoEstimadoCriacao(?vmh, ?criacao) ∧
15       swrlb:lessThan(?paginas, ?tamSF)   ∧
16       swrlb:subtract(?diferenca, ?custoReal, ?paginas)   ∧
17       swrlb:multiply(?multiplicado, ?diferenca, 100)   ∧
18       swrlb:divide(?porcentagem, ?multiplicado, ?custoReal)   ∧
19       swrlb:greaterThan(?porcentagem, 50)   ∧
20       swrlx:createOWLThing(?vmr, ?vmh)  ∧
21       swrlb:subtract(?restanteSF, ?tamSF, ?paginas)   ∧
22 Heuristica(?heuristica) ∧
23       temNome(?heuristica,
24 "HeuristicaVisaoMaterializadaExpectativa") →
25 VisaoMaterializadaReal(?vmr) ∧
26       eGeradoPor(?vmh, ?vmr) ∧
27       temNome(?vmr, ?nomeVmh) ∧
28       temValorBonusAcumulado(?vmr, 0) ∧
29       temDescricao(?vmr, ?descricao) ∧
30       temTamanhoDiscoSF(?bd, ?restanteSF) ∧
31       temSituacao(?vmr, "criada") ∧
32       criadaPor(?vmr, ?heuristica)
```

Figura 4.11: Regra SWRL usada para inferir visões materializadas reais através da heurística de expectativas (HExp)

Nas linhas 1 até 14 são verificadas as existências dos indivíduos necessários para a execução da HExp. As linhas 15 a 19 executam a regra criada a partir da hipótese da heurística de expectativas. Caso todas as condições da regra sejam satisfeitas, as linhas 25 até 32 realizam a instanciação de uma visão materializada real e seus atributos com os respectivos valores.

4.2.5
Persistência das visões materializadas reais

Após as visões materializadas reais serem instanciadas, o próximo passo é persisti-las no banco de dados. A Figura 4.12 exibe o comando SWRL que consulta quais VMRs foram inferidas pelas heurísticas da ontologia.

```
1  VisaoMaterializadaReal(?vmr)   ∧
2       temSituacao(?vmr, ?situacao)   ∧
3       temNome(?vmr, ?nome)   ∧
4       eGeradoPor(?vmh, ?vmr)   ∧
5       temValorBonusAcumulado(?vmh, ?bonus)   ∧
6       temValorCustoEstimadoCriacao(?vmh, ?custoCriacao)∧
7       temDescricao(?vmr, ?comando)   →
8   sqwrl:select(?vmr,   ?nome,   ?bonus,   ?custoCriacao,
9  ?comando, ?situacao) ∧ sqwrl:orderBy(?custoCriacao)
```

Figura 4.12: Consulta SQWRL usada para recuperar visões materializadas reais inferidas

A consulta obtém as visões materializadas reais que foram instanciadas pelas heurísticas de benefícios, expectativas ou outra heurística qualquer. O resultado da consulta SWRL (Figura 4.10) é o resultado de todo o processo de seleção de visões materializadas, são as ações de sintonia fina que, caso persistidas no banco de dados, possibilitarão que a carga de trabalho possa usufruir de seu ganho.

4.2.6
Conclusões

As extensões realizadas nas ontologias de domínio e de tarefas serviram para experimentar e documentar o processo de extensão da heurística de sintonia fina. Durante a extensão da ontologia de domínio, foi necessário criar novos conceitos, relacionamentos e atributos. Porém o processo não teve um grande impacto nos conceitos e regras já existentes. Com exceção de um único atributo renomeado, não foram necessárias intervenções que afetassem as heurísticas já existentes na ontologia.

Isso corrobora a viabilidade de se realizar sintonia fina através de ontologias e, além disso, a possibilidade de coexistência de heurísticas para múltiplos tipos de ações de sintonia fina e múltiplas heurísticas para um único tipo de ação.

Foram mostrados neste capítulo os conceitos, atributos e relacionamentos criados na ontologia de domínio (Seção 4.1), a extensão da ontologia de tarefas

com a inclusão das heurísticas: (i) heurística de visões materializadas hipotéticas (Seção 4.2.1); (ii) heurística de benefícios (Seção 4.2.2); e heurística de expectativas (Seção 4.2.4). O Capítulo 5 será apresentado o projeto de arquitetura proposto para o *framework* Outer-Tuning.

5
Framework Outer-Tuning

Como uma das contribuições dessa pesquisa, foi desenvolvido o projeto de arquitetura e a implementação do *framework* Outer-Tuning. Na Seção 5.1 estão descritos os requisitos funcionais do *framework* Outer-Tuning, seguida pela Seção 5.2 que propõe uma arquitetura de software que atende a estes requisitos e a Seção 5.3 com as conclusões do capítulo.

5.1
Requisitos funcionais do *framework* Outer-Tuning

Durante a definição do escopo do *framework* Outer-Tuning (Almeida13), levantaram-se requisitos desejáveis. Entre eles, os requisitos de 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, foi desenvolvido o projeto de arquitetura para o *framework*.

5.2 Projeto de arquitetura

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 5.1). A partir daí, criou-se o projeto de arquitetura do *framework* Outer-Tuning (Figura 5.2), que foi seguido de sua implementação.

5.2.1 Fases de execução do *framework* Outer-Tuning

Uma das decisões importantes tomadas para definir as fases de execução (Figura 5.1), foi decidir ***em qual momento*** e ***como*** a máquina de regras seria utilizada. Uma máquina de regras é o componente pelo qual as regras de inferência das heurísticas (definidas na ontologia de tarefas) são selecionadas e executadas.

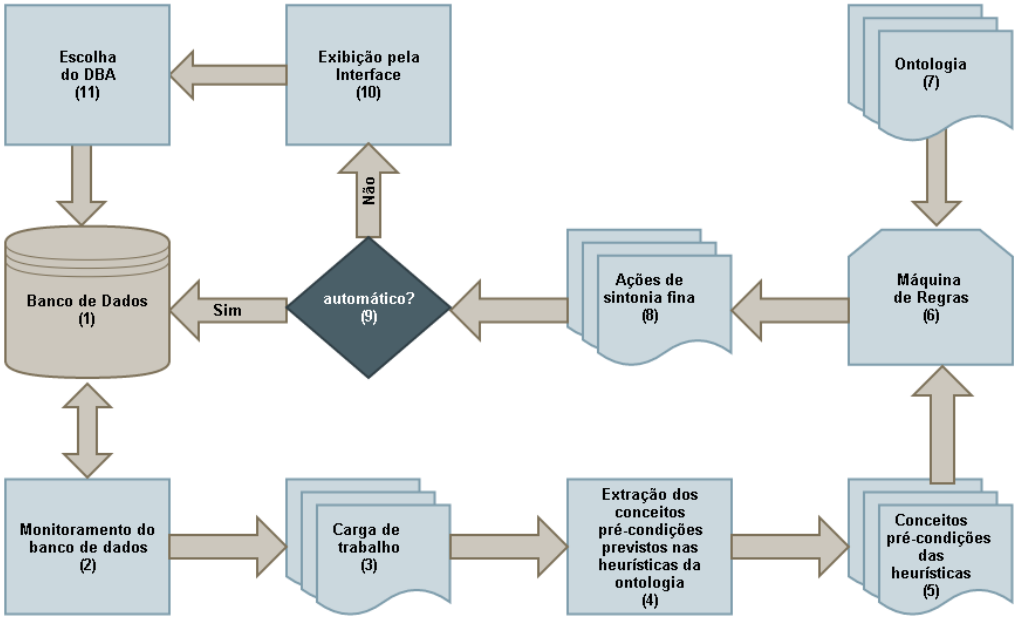

Figura 5.1: Fluxo de execução projetado para o framework Outer-Tuning

O fluxo definido para o Outer-Tuning (Figura 5.1) se inicia com o monitoramento do banco de dados de forma não intrusiva e contínua (2), que captura a carga de trabalho (3) submetida no banco de dados (1) por seus usuários. Em seguida são extraídos (4) da carga de trabalho os conceitos pré-condições das heurísticas contidas na ontologia (5). Depois de extraídos, eles são instanciados na máquina de regras (6), que também recebe 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 um processo de sintonia fina automática (9), serão executadas todas as ações de sintonia fina 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 soluções deseja executar no banco de dados (1).

5.2.2 Projeto de arquitetura 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 (Brown96). Foi escolhida para o Outer-Tuning uma arquitetura baseada em componentes 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 5.1) independentes. Caso necessário, os componentes poderiam ser substituídos/mantidos de forma compartimentalizada sem a propagação de *bugs*. A comunicação entre os componentes foi feita através de interfaces bem definidas, permitindo a independência e a capacidade de serem substituídos. Além dessas vantagens, componentes são umas das formas de reuso de software mais viáveis e usadas (Sommerville2011).

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 organizada de componentes de software reusáveis (Sommerville2011). A escolha desse tipo de *framework*, assim como o uso de componentes, foi feita para atender os requisitos funcionais (Seção 5.1), para dar qualidade ao software, possibilitar uma futura evolução para software orientado a serviços (Azevedo09), é principalmente permitir um baixo acoplamento entre as partes do software.

Uma outra característica do uso de componentes para a arquitetura do *framework* é a definição clara das partes *cold* e *hot spots*. Considera-se *cold spot* uma parte do *framework* com um alto acoplamento e grande impacto para extensão. Já *hot spot* uma parte que possui um baixo acoplamento e, consequentemente, de fácil extensão, sem um impacto no funcionamento do *framework*.

De acordo com o fluxo de execução proposto (Figura 5.1), o Outer-Tuning teve seus componentes definidos e especializados para cada etapa do ciclo. Na Figura 5.2, são enumerados e a seguir 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

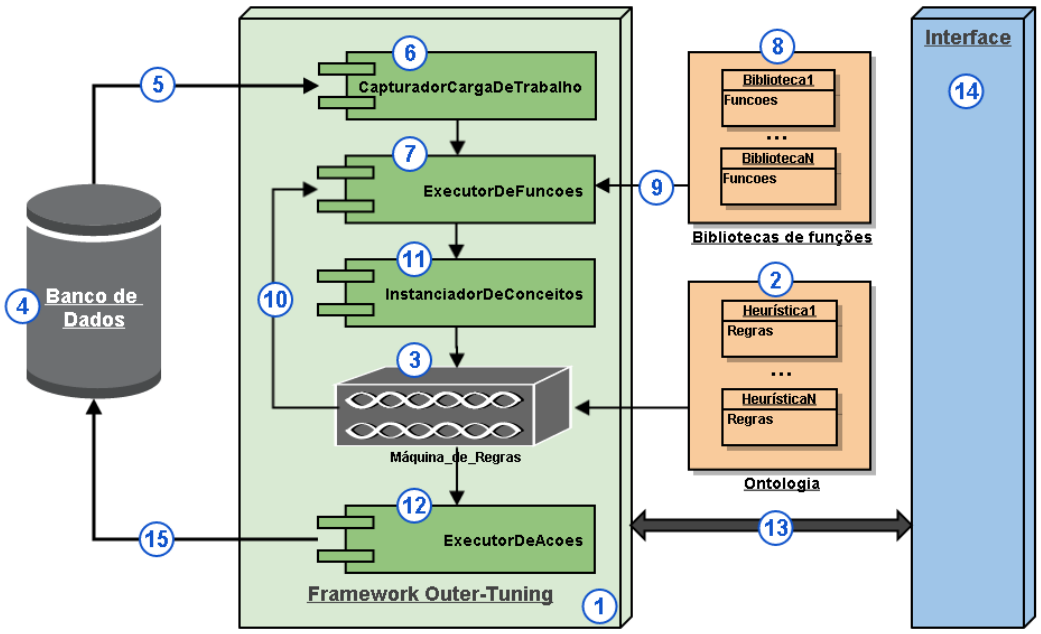

Figura 5.2: Projeto de Arquitetura do *framework* Outer-Tuning - enumerado

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*). 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**: 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 (Friedman14). Sua escolha se deu pela compatibilidade entre as interfaces e os requisitos. O requisito funcional (iv) (Seção 5.1) 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 uma funcionalidade decisiva para que o DBA, através da interface do

Outer-Tuning, possa 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 relacional.

(5) **Carga de trabalho**: A carga de trabalho capturada consiste em comandos SQL do tipo DML, 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 (Oracle2014). 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 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. Este componente possui três fontes de entrada de dados:

   (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. Cada heurística possui conceitos que são pré-requisitos para a sua execução, e as funções tem o objetivo de gerar instâncias desses conceitos com base na carga de trabalho capturada.
   (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 a 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 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, já 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 heurísticas. 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. Há compartilhamento do sistema de conexão com o banco de dados entre o componente *ExecutorDeAcoes* e 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)(Khosla04) pela possibilidade da execução sem nenhuma interface e para que a interface seja totalmente independente do *framework*. Com a utilização do padrão quadro-negro, a interface durante os testes e implementação pôde ser substituída por um arquivo de configurações, que pode facilitar a depuração do código-fonte 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 a selecionar quais heurísticas deseja executar (Figura 5.3); (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 sintonia fina**: 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.

5.2.3 Projeto de Interface

A interface (vide (4) na Figura 5.2), foi projetada de forma a ser um componente independente do *framework*. O método de comunicação entre os componentes escolhido foi o quadro-negro. Um agente inicia uma ação de comunicação escrevendo um item de informação no quadro. Esta informação é

então disponível para todos os outros agentes no sistema. Todo agente pode, em qualquer tempo acessar o quadro, para ver se alguma nova informação está disponível desde seu último acesso. Se sim, ele pode ler essa informação.

No exemplo concreto do *framework* Outer-Tuning e sua interface, eles compartilham um espaço em disco em que podem transmitir informações através de arquivos texto. Nesse diretório o *framework* cria arquivos (distintos para cada execução) das informações: heurísticas disponíveis na ontologia, carga de trabalho, ações de sintonia fina, e log de execução. Por sua vez, a interface escreve quais heurísticas o DBA escolheu, o modo de execução escolhido (automático ou semiautomático), e quais ações de sintonia fina escolhidas para serem persistidas no banco de dados.

A captura da carga de trabalho pode gerar muitos dados para serem exibidos de uma única vez ao usuário. Com ele também se conseguiu uma liberdade para que a interface leia apenas o recorte de dados que lhe interessa. Exemplo: o DBA pode escolher visualizar apenas os últimos 10 comandos capturados, ou ainda, os 10 comandos com maior frequência de captura. Feito essa escolha, a ferramenta consegue ler apenas o trecho do arquivo texto que corresponde aos filtros escolhidos sem trocar nenhum pedido com o *framework*, e sem ter que ler todos os comandos capturados do arquivo texto.

Com o nível de independência projetado, foi implementada uma interface web para a visualização dos dados de saída do *framework*. Isso implica que futuramente, quando houver um projeto de interface com um *design* refinado e com estudos de usabilidade (fora do escopo dessa pesquisa), a interface poderia ser desenvolvida utilizando outras tecnologias e substituída sem nenhuma intervenção no código ou fluxo de execução do *framework*.

A interface atual foi projetada para atender às funcionalidades principais do *framework*, e para que fosse possível testar o *framework* tendo uma perspectiva das capacidades da ferramenta. Foram considerados os seguintes funcionalidades pela interface:

- Iniciar a execução do *framework*, para que ele leia da ontologia quais heurísticas estão disponíveis para a atividade de sintonia fina;
- Escolha de quais heurísticas o DBA deseja executar;
- Escolha entre o modo automático e semiautomático;
- Visualização da carga de trabalho capturada;
- Visualização da justificativa para cada ação de sintonia fina inferida pelo *framework*;
- Escolha de quais ações o DBA deseja efetivamente persistir no banco de dados;

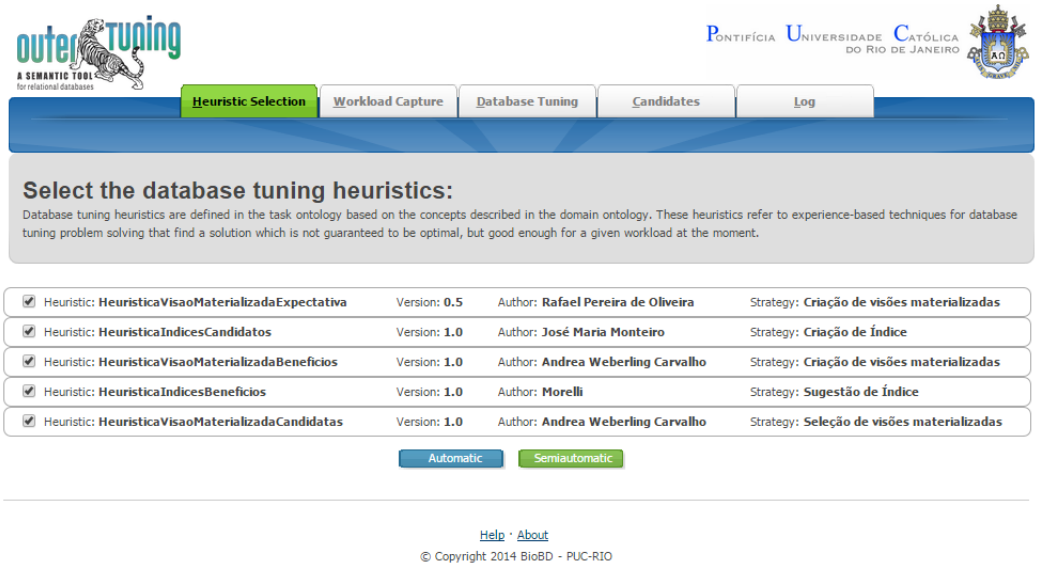

Figura 5.3: Tela para seleção de heurísticas

- Visualização do log de execução do *framework*;

Portanto, para realizar a escolha entre quais ações de sintonia fina deseja executar, o DBA terá informações sobre a composição da carga de trabalho, estatísticas de execução e os ganhos e custos hipotéticos calculados para cada ação inferida pelas heurísticas.

Tela para seleção de heurísticas

A tela de seleção de heurísticas Figura 5.3 é a primeira a ser acionada. Nela, o DBA pode escolher quais heurísticas deseja executar durante a atividade de sintonia fina. Ela mostra informações sobre as heurísticas contidas na ontologia: o nome da heurística, a versão, o autor e a estratégia abordada pela heurística.

Após a escolha da(s) heurística(s), o DBA deve então dar início à ferramenta de sintonia fina, através do modo automático e semiautomático. A principal diferença entre eles, é que no modo automático o *framework* não consulta quais ações de sintonia fina (dentre as inferidas pelas heurísticas escolhidas) o DBA deseja aplicar e as executa todas automaticamente no banco de dados. Já na execução semiautomática o DBA precisa escolher quais ações inferidas o *framework* deve consumar.

Tela para exibição da carga de trabalho capturada

Essa tela tem a função de exibir a carga de trabalho capturada. Nela são mostrados os N comandos DML com o maior número de reincidência na carga de trabalho. O número N de comandos mostrados pode ser escolhido através do arquivo de configuração do *framework*.

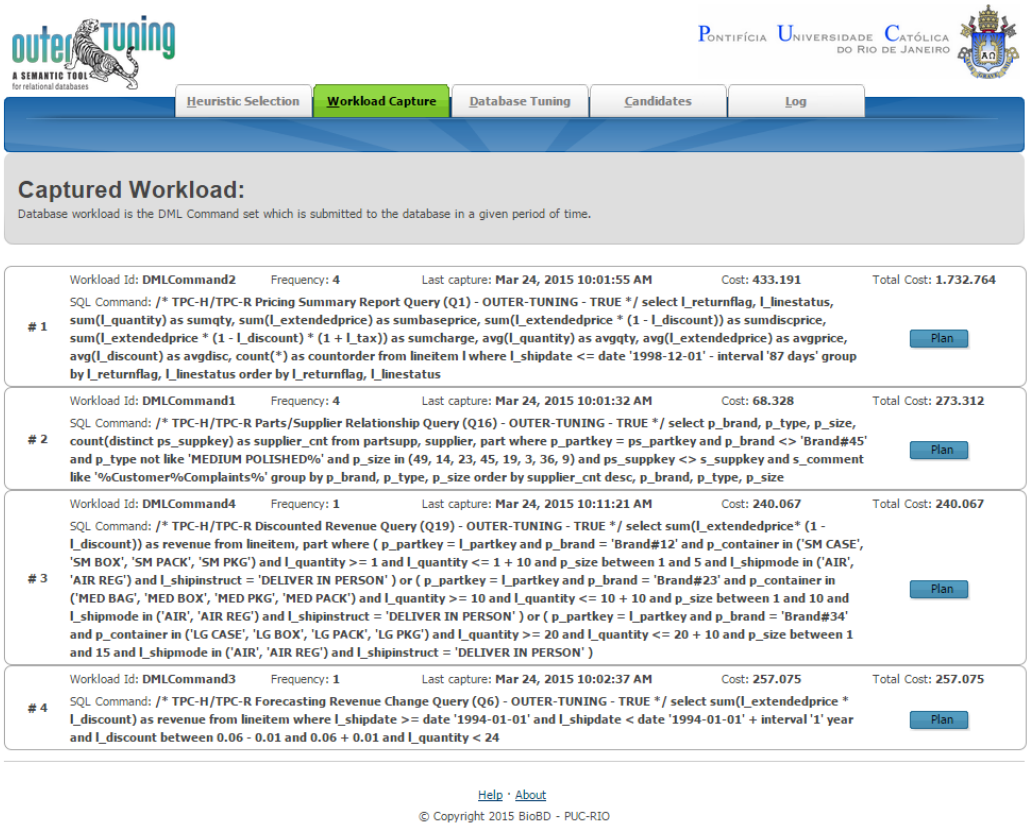

Figura 5.4: Tela para exibição da carga de trabalho capturada

A tela exibe informações sobre os comandos DML capturados da carga de trabalhos, entre elas a quantidade de capturas, a data da última captura, o custo de execução, o custo total que é a quantidade de capturas multiplicado por seu custo.

Essa tela também possui a opção de expandir a informação de cada comando, visualizando o seu plano de execução e o momento de cada captura:

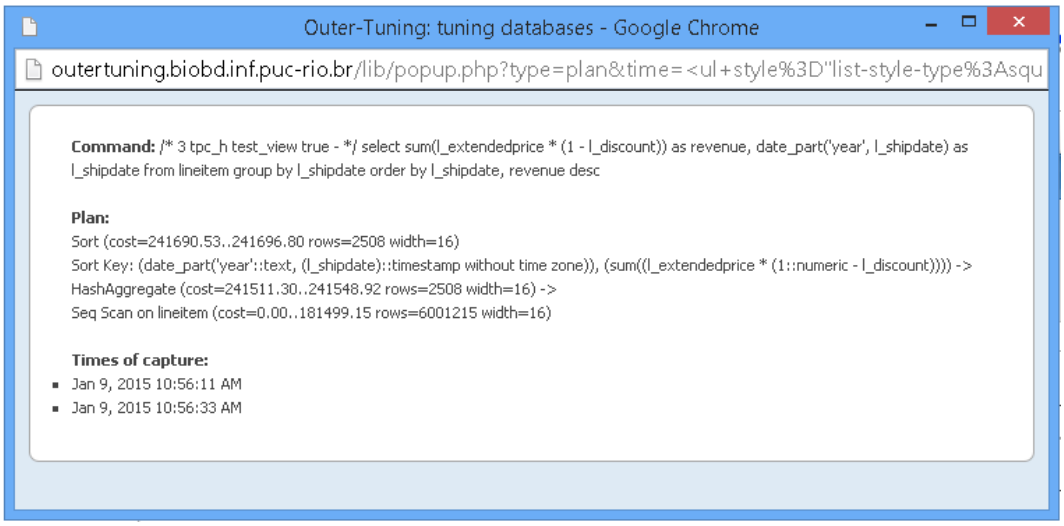

Figura 5.5: Interface - Tela para exibição de detalhes da carga de trabalho capturada

Tela para seleção de ações de sintonia fina inferidas

Nessa tela, o DBA pode visualizar as ações inferidas pelas heurísticas de sintonia fina escolhidas no início do processo, e escolher quais ações o *framework* deve executar. Também é possível visualizar uma justificativa para cada ação através do botão "*justify*". Ao acionar esse comando a tela abaixo é exibida, contendo uma justificativa para a sua escolha como uma ação de sintonia fina que trazer algum benefício para a carga de trabalho executada no banco de dados.

Figura 5.6: Interface - Tela para seleção de ações de sintonia fina

Através da justificativa, o DBA pode ter informações de como a ação de sintonia fina foi inferida. Isso ajudará na escolha de qual ação ele deve priorizar, e também em relatórios que exijam uma justificativa para as ações escolhidas.

Tela para exibição do log do framework

Nessa tela é mostrado o log operacional do *framework*. Nele existem informações sobre a captura da carga de trabalho, as instâncias geradas pela execução das funções, o estado da máquina de regras e ações de sintonia fina escolhidas pelo DBA. Por exemplo, a **linha 06:31:42** apresenta informação que a ação de criação da visão materializada *public.v_ot_workload_1* teve início naquele horário.

O objetivo do Log é exibir informações sobre o funcionamento do *framework*, os passos de execução da inferência de ações de sintonia fina, as escolhas realizadas pelo DBA e tornar mais próxima a execução do *framework* do resultado exibido pelas outras telas.

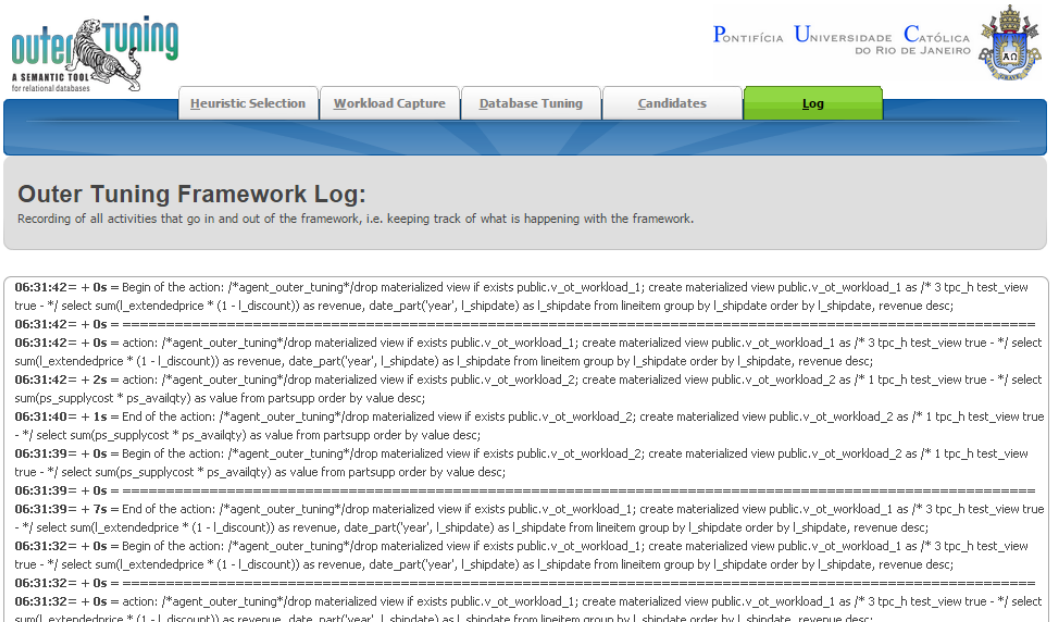

Figura 5.7: Interface - Tela de Log

5.3 Conclusões

Foi apresentado aqui o projeto de arquitetura proposto para o *framework* Outer-Tuning. Foi apresentado um fluxo de execução, e com base neste fluxo, foi proposta e descrita a arquitetura baseada em componentes. Também foi mostrado o projeto de interface web para o *framewok*.

O Capítulo 6 apresenta resultados experimentais da execução do Outer-Tuning. Foi realizada uma avaliação experimental através da execução de um *benchmark*, os resultados da execução das heurísticas de seleção e criação de visões materializadas foram comparados e analisados.

6
Resultados experimentais

O *framework* Outer-Tuning se propõe a justificar ações de sintonia fina através da ontologia de aplicação e compará-las para apoiar o DBA na tarefa de sintonia fina. O método encontrado para validar essa proposta foi, além de efetivamente implementar o *framework*, comparar duas heurísticas de sintonia fina para seleção e criação de visões materializadas. Através da execução da heurística de benefícios e da heurística de expectativas, compararam-se as ações de sintonia fina inferidas por ambas através das informações obtidas pela ferramenta.

A qualidade das ações de sintonia fina está relacionada diretamente com a qualidade da heurística e sua compatibilidade com a carga de trabalho. Independentemente do desempenho de ambas as heurísticas durante os testes, o resultado apresentado mostra que a ontologia, através da extensão realizada, consegue propor a criação de visões materializadas e que o Outer-Tuning é capaz de executar simultaneamente mais de uma heurística para um mesmo tipo de sintonia fina e fornecer informações para a comparação das ações inferidas.

Cabe lembrar que não é nosso objetivo obter os melhores resultados ou as melhores heurísticas, mas sim, poder estudar e avaliar as alternativas de sintonia fina existentes.

6.1
Ambiente de testes e pressupostos

Foi utilizado um subconjunto de consultas (vide Anexos) do *benchmark* TPC-H que possui uma carga de trabalho analítica (OLAP). A carga de trabalho foi utilizada com o SGBD PostgreSQL na versão 9.1 e sistema operacional Ubuntu 12.04 32bit, instalado em uma máquina virtual configurada para ter um processador 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 é considerada benéfica quando as consultas reescritas e forçadas a utilizarem a VM trazem um ganho para a execução da carga de trabalho. Já as maléficas são as VMs que trazem perdas para a execução da carga de trabalho quando são consideradas.

Cabe observar que o PostgreSQL não utiliza visões materializadas durante a criação do plano de execução de uma consulta (seção 6.4.4). Isso significa que, mesmo que uma VM tenha sido criada para responder a uma consulta específica, o PostgreSQL não utilizará a VM de forma automática. Durante os testes foi

necessário forçar o SGBD a utilizar as VMs criadas através da reescrita das consultas. A reescrita consistiu em substituir as referências às tabelas originais da consulta pelas VMs criadas pelo Outer-Tuning. Todas as consultas originais e reescritas estão detalhadas nos apêndices.

6.2 Comparação das heurísticas de criação de visões materializadas

Como já descrito, foram inseridas na ontologia de sintonia fina três heurísticas: uma heurística de seleção de visões materializadas hipotéticas responsável por criar visões materializadas hipotéticas para cada uma das consultas da carga de trabalho, e duas heurísticas para a seleção (dentre as VMs hipotéticas) de quais VMs devem ser criadas fisicamente no banco de dados: a heurística de benefícios e a heurística de expectativa.

Como resultado da execução do Outer-Tuning, a heurística de benefícios propôs a criação de 5 VMs e a heurística de expectativas propôs 9 VMs. As heurísticas tiveram um ganho total na execução da carga de trabalho equivalente (Figura 6.1). O ganho total foi calculado pela soma do tempo de execução de todas as consultas da carga de trabalho utilizando as VMs sugeridas pelas heurísticas. A heurística de benefícios teve um ganho total de 12,4% no tempo de execução da carga de trabalho. Já a heurística de expectativas teve um ganho de 12,2%.

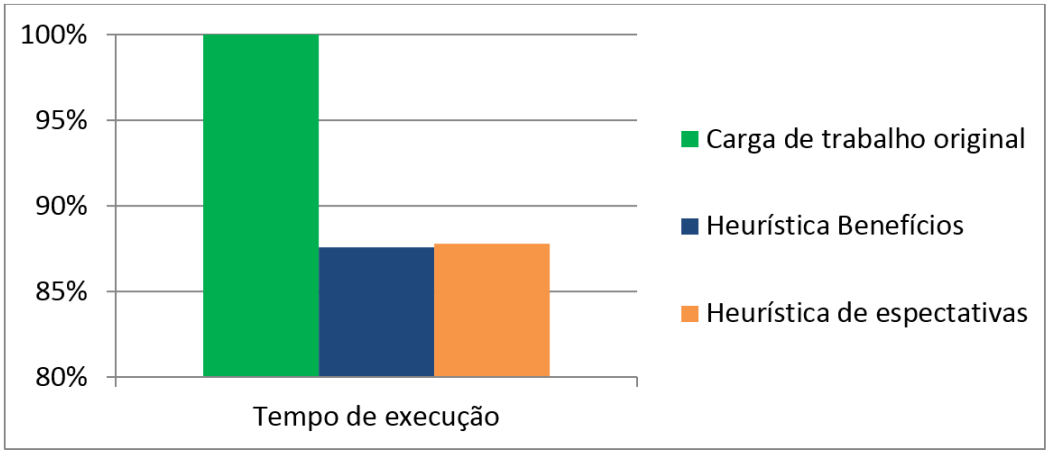

Figura 6.1: Custo total de cada heurística em relação à carga de trabalho original

Uma diferença evidenciada entre as heurísticas pelo Outer-Tuning foi 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 que incluem todas as selecionadas pela heurística de benefícios. Logo, o custo de armazenamento e de criação da heurística de expectativas foi maior.

Durante a execução do Outer-Tuning, a heurística de benefícios também se diferenciou da heurística de expectativas em relação ao tempo necessário para a inferência das VMs propostas. A heurística de benefícios analisa a carga de trabalho continuamente e espera até que o ganho estimado acumulado de uma visão materializada hipotética seja maior que o seu custo de criação estimado. Nos casos em que uma consulta possui um custo de criação estimado muito alto, a consulta necessita ser capturada várias vezes até que a soma do ganho hipotético seja maior que o custo de criação. Já a heurística de expectativas não considera a frequência do comando na carga de trabalho para a seleção de uma visão materializada, após uma única captura a heurística pode decidir ou não criar a VM. A consequência disso durante os testes foi que a heurística de benefícios teve que ser executada por várias horas para propor as suas ações de sintonia. Já a heurística de expectativas levou apenas o tempo necessário (cerca de meia hora) para o Outer-Tuning capturar, pelo menos uma vez, todas as consultas da carga de trabalho.

As 5 VMs propostas pela heurística de benefícios foram baseadas nas consultas Q1, Q4, Q5, Q9 e Q12 do TPC-H. As VMs baseadas nas consultas Q1, Q5 e Q9 trouxeram um ganho médio positivo para a carga de trabalho. Já as visões baseadas nas consultas Q4 e Q12 trouxeram um prejuízo para a carga de trabalho (Figura 6.2).

VMs benéficas	Ganho (%)
Q01	99,9%
Q05	98,1%
Q09	77,3%

VMs maléficas	Ganho (%)
Q04	-6,1%
Q12	-27,4%

Figura 6.2: Visões sugeridas pela heurística de benefícios

A heurística de expectativas criou 9 VMs baseadas nas consultas Q01, Q03, Q04, Q05, Q06, Q07, Q09, Q12, Q14. As VMs baseadas nas consultas Q01, Q05, Q06 e Q09 trouxeram ganhos em sua execução. Porém, as VMs baseadas nas consultas Q03, Q04, Q07, Q12 e Q14 trouxeram prejuízos.

VMs benéficas	Ganho (%)
Q01	99,9%
Q05	98,2%
Q06	86,1%
Q09	74,3%

VMs maléficas	Ganho (%)
Q03	-3,0%
Q04	-3,6%
Q07	-43,2%
Q12	-39,6%
Q14	-6,3%

Figura 6.3: Visões sugeridas pela heurística de expectativas

A heurística de expectativas inferiu quatro VMs a mais que a heurística de benefícios. Dentre elas, três trouxeram prejuízos (Q03, Q04 e Q14). Mas também

inferiu um VM ignorada pela heurística de benefícios (Q06) que trouxe 86,1% de ganho na execução de sua consulta original. Na execução total da carga de trabalho, a descoberta pela heurística de expectativas da VM positiva baseada na consulta Q06 compensou as perdas das visões que trouxeram prejuízos e esta é a razão da equivalência de ganhos entre as duas heurísticas (12,4% e 12,2%).

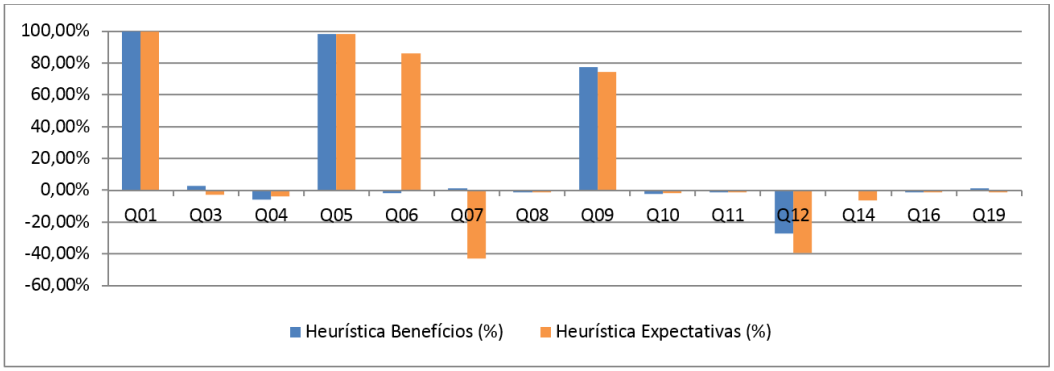

Figura 6.4: Ganho (em %) no tempo de execução das consultas na presença de visões materializadas sugeridas pelas heurísticas de benefícios e de expectativas.

A diferença de resultados na execução das duas heurísticas se deu pela diferença nas estratégias de cada uma. A heurística de benefícios utiliza a frequência das consultas na carga de trabalho para considerar uma VM útil ou não, enquanto a heurística de expectativas é uma proposta menos restritiva e não considera a frequência de uma consulta. Isso fez com que a heurística de expectativas selecionasse um conjunto maior de VMs, já que ela não considera todas as execuções da carga de trabalho, como faz a heurística de benefícios.

6.3 Justificativas de sintonia fina

O Outer-Tuning é capaz de propor justificativas para as ações de sintonia fina inferidas. Quando uma regra SWRL instancia uma visão materializada hipotética ou real na máquina de regras, é criada uma justificativa para esta ação de sintonia fina a partir do contexto que foi criada a VM.

As justificativas são possíveis graças à semântica adicionada ao processo de sintonia fina pelo uso de ontologias. Através das definições da ontologia de domínio, é possível criar justificativas para entender, dentre todas as possibilidades analisadas pelas heurísticas, quais informações da carga de trabalho levaram o Outer-Tuning a propor aquelas ações.

Atualmente as justificativas são criadas a partir de modelos armazenados na ontologia de sintonia fina. Cada heurística possui um modelo que é instanciado de acordo com os parâmetros utilizados pelas regras SWRL para realizar a inferência

da ação. Este modelo, assim como toda a ontologia, pode ser adequado ou estendido para propor justificativas mais complexas de acordo com as necessidades do DBA.

Para cada heurística incluída na ontologia durante esta pesquisa (Capítulo 4), foi criado um template de justificativa. A heurística de seleção de visões materializadas hipotéticas é responsável por criar uma visão materializada hipotética para cada consulta capturada no banco de dados. Todas as visões materializadas hipotéticas possuem o seguinte template de justificativa:

```
1 A consulta SQL:
2
3 <Consulta>
4
5 foi reescrita para torná-la genérica e passível de uso por outras
      consultas com a mesma estrutura, mas parâmetros diferentes.
6
7 Como consequência, foi sugerida a visão materializada hipotética
     com a seguinte sintaxe:
8
9 <DLL Create View>
```

A heurística de benefícios é responsável por selecionar quais visões materializadas hipotéticas devem ser criadas fisicamente no banco de dados (Capítulo 4). As suas justificativas possuem o seguinte template:

```
1 Esta visão materializada possui o benefício estimado em
2 <VisaoMaterializada.beneficioAcumulado>, o que supera o
3 seu custo de criação estimado de
4 <VisaoMaterializada.custoCriacao>. Comandos que geraram os
5 benefícios:
6
7 <VisaoMaterializada.comandosBeneficiados>
```

Por fim, a heurística de expectativas, assim como a de benefícios, é responsável por selecionar quais visões materializadas hipotéticas devem ser criadas fisicamente (seção 4.2.4). O seu template de justificativa é o seguinte:

```
1 Esta visão materializada possui o custo estimado de
2 execução de <VisaoMaterializada.custoEstimadoExecucao>,
3 enquanto a consulta original possui o custo de
4 <Consulta.custoExecucao>. Logo, a economia gerada a cada
5 utilização da visão materializada será de
6 <VisaoMaterializada.ganhoExecucao>
```

Os templates apresentados ilustram a possibilidade de gerar justificativas a partir da ontologia para as ações de sintonia fina. Todas as ações de sintonia fina são baseadas na carga de trabalho instanciada na ontologia. Logo, é possível exibir ao

usuário qualquer informação referente ao processo de sintonia fina através desses templates.

6.4 Análise dos resultados e discussão

Descreve-se aqui a análise dos resultados e pontos observados durante a pesquisa que foram considerados relevantes para a discussão sobre sintonia fina, e o trabalho apresentado.

6.4.1 Visões materializadas podem trazer benefícios ou prejuízos

Os ganhos na utilização de VMs por parte das consultas foram decorrentes, principalmente, da sumarização e o armazenamento em ordem dos dados. Se considerarmos o exemplo da consulta Q01 podemos exemplificar esse fato. A consulta Q01 foi projetada pelo TPC-H para sumarizar preços de produtos e quantidades de pedidos. A tabela *lineitem* usada para a execução da consulta Q01 não possui índice secundário em nenhuma coluna, e para executar a consulta original o SGBD precisa realizar uma leitura completa (*fullscan*) na tabela *lineitem*, filtrá-los, agrupá-los e ordená-los . Segundo o SGBD, o custo total da operação é de 431.832. Já para executar sobre a VM, considerando todas as operações citadas, o custo total é de 198. Podemos observar na tabela abaixo que é um custo 99,95% menor. Essa análise permite visualizarmos a relação entre quantidade de páginas lidas e processadas, com o custo total de execução de uma consulta.

Atributo	**Original Q01**	**VM Q01**	**Ganho em utilizar a VM (%)**
Custo real total	431.832	198	99,95%
Páginas lidas	120.013	57	99,94%

Tabela 6.1: Comparação entre custos de execução da consulta Q01 E VM Q01

Todas as outras VMs criadas seguem o mesmo raciocínio: Se a VM criada tiver uma quantidade de páginas menor do que a quantidade de páginas lidas para executar a consulta original, ela necessariamente trará um ganho na execução da carga de trabalho. Da mesma forma se considera as visões materializadas que trouxeram prejuízos. A Q07 foram uma das que trouxeram prejuízos. Se observarmos a quantidade de páginas geradas podemos ver a mesma relação:

Atributo	**Original Q07**	**VM Q07**	**Ganho em utilizar a VM (%)**
Custo real total	88.716	237.432	-167,63%
Páginas lidas	39.848.067	720.145.800	-1.707,23%

Tabela 6.2: Comparação entre custos de execução da consulta Q07 E VM Q07

A princípio, a ideia de gravar no disco o resultado de uma consulta para usá-lo posteriormente não nos faz imaginar que ler o resultado processado e ordenado terá um custo maior do que executar a consulta original. Porém, esta situação ocorre devido à generalização da consulta explicada na Seção 6.4.2. Quando reescrevemos uma consulta para gerar uma visão materializada, são removidos os parâmetros de restrição da cláusula *where* da consulta. Isso deixa o resultado (em páginas) do comando que gerará a visão materializada maior do que o resultado da consulta original. Essa diferença, foi o que tornou a VM usada como exemplo acima, inviável para a execução da consulta original.

Através do Outer-Tuning o DBA, após as devidas comparações, 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 destas na ontologia, e inclusão de novas heurísticas próprias ou da literatura (Capítulo 4). Nada impede, por exemplo, que o DBA realize uma combinação entre as heurísticas de benefícios e expectativas criando a sua própria heurística capaz de selecionar VMs positivas como a VM baseada na Q06 inferida pela heurística de expectativas, mas evitando criar as VMs negativas Q7, Q3, Q14 como fez a heurística de benefícios.

6.4.2 Consultas candidatas para visões materializadas

Consideram-se boas candidatas para materialização as consultas que possuem o tamanho do resultado retornado (em páginas) menor que o total de páginas lidas durante a sua execução. Em contraposição a essas, consideram-se escolhas ruins para a criação de VMs as consultas que realizam apenas varreduras em tabelas da consulta. Segue um exemplo de uma consulta que pode ser considerada parte se tornar uma VM:

```
1  select
2      sum(valor_produto * quantidade) as valor,
3      ano_venda,
4      vendedor_id
5  from
6      vendas
7  group by
8      ano_venda,
9      vendedor_id
10 order by
11     ano_venda
```

Essa consulta sumariza o valor das vendas (linha 2) de todos os anos (linha 3) da tabela vendas (linha 6) agrupando-os pelo ano da venda (linha 8) e pelo vendedor

(linha 9). Caso essa consulta seja materializada como uma VM pode-se saber o valor total das vendas de cada vendedor de acordo com o ano. Esse exemplo foi comprovado através de testes manuais como uma boa alternativa para a criação de VMs.

Como exemplo de consultas candidatas ruins pra materialização, podemos utilizar o exemplo abaixo, que realiza apenas uma varredura na tabela vendas do banco de dados:

```
1 select
2     *
3 from
4     vendas
5 where
6     ano_venda = '2014'
```

Consultas como esta, caso escolhidas como VM, irão apenas duplicar o conteúdo da tabela vendas no disco rígido. O custo de executar essa consulta utilizando a visão materializada será no máximo igual ao custo de executar utilizando a tabela vendas. A quantidade de páginas lidas será a mesma, já que não há presença de agregação e funções de cálculo.

Como já descrito, foi utilizado o benchmarck TPC-H para testar o Outer-Tuning. Mas durante a fase de implementação do *framework* foi utilizada também uma carga de dados sintética, criada para evidenciar as duas características acima mencionadas. De fato, foram criadas seis consultas que realizavam sumarizações sem nenhuma restrição na cláusula *where* e outras seis consultas que realizavam apenas varreduras em tabelas, também sem nenhuma restrição na cláusula *where*.

Uma das consultas que propositalmente não traria ganhos para a carga de trabalho considerada, caso se optasse por criar uma VM, foi a seguinte:

```
1 select
2     o_orderkey ,
3     o_custkey ,
4     o_comment
5 from
6     orders
```

Teoricamente, essa consulta teria o mesmo desempenho se executada diretamente ou substituindo a tabela pela VM. Porém o Outer-Tuning apontou a visão materializada candidata baseada nessa consulta como benéfica. A princípio parecia um erro nas regras SWRL que definem a heurística de benefícios. Entretanto, após realizarmos um processo de *debug* (que não apontou erro algum), decidiu-se criar a VM no banco de dados e verificar as consequências. Como resultado do teste, foi constatado que a heurística estava certa e que realmente,

a execução da consulta utilizando a VM era benéfica. Seguem os custos reais de execução da consulta:

Consulta original	**Consulta utilizando VM**
Seq Scan on orders (cost=0.00..41453.00 rows=1500000 width=61) (actual time=0.021..3582.704 rows=1500000 loops=1)	Seq Scan on v_ot_workload_q1 (cost=0.00..31807.00 rows=1500000 width=61) (actual time=0.027..3414.035 rows=1500000 loops=1)
Total runtime: 6808.295 ms	**Total runtime: 6422.749 ms**

Tabela 6.3: Planos de execução de consulta sintética exemplo

A explicação para este ocorrido é que, como a VM é um recorte da tabela, a VM possui um número de páginas menor do que as páginas lidas durante a execução da consulta na tabela *orders*. Além disso, contribui para o fato que a VM tenha sido populada em uma única operação e as páginas ficaram gravadas em sequência configurando uma clusterização. Portanto, em casos especiais como este, onde há um recorte de uma tabela maior, uma VM mesmo que não contendo em sua definição sumarizações ou ordenação pode trazer ganhos para a carga de trabalho.

6.4.3 Estatísticas desatualizadas

Ao trabalhar com números absolutos, como quantidade de páginas no disco e quantidade de linhas de uma tabela ou custo de execução, sempre há uma margem de erro inerente à forma que estes são calculados.

As heurísticas de expectativas e benefícios se apoiam em custos baseadas no plano de execução do SGBD. Entretanto, um plano é uma estimativa baseada nas estatísticas do banco de dados. Durante os testes, na conferência manual dos resultados foi detectado um exemplo concreto de que as estimativas podem ter variações em relação aos valores reais: o Outer-Tuning extraiu do plano de execução da consulta Q11 um número de tuplas igual à 193.034. Porém ao se executar a consulta, o número real era de 200.000 linhas. A diferença foi de 6.966 tuplas – um erro de 3,5%.

Erros nas estatísticas podem criar ações de sintonia fina que trazem prejuízo à carga de trabalho. A diferença entre estimativas do plano de execução e os valores reais significa que deve-se considerar a margem de erro no cálculo dos benefícios e malefícios antes de se optar por uma ação de sintonia fina.

Quando a heurística de expectativas foi definida, utilizou-se no parâmetro X% (Regra da Figura 4.11), que é a margem de ganho mínimo para que uma VM seja materializada o valor de 50%. Durante os testes, a consulta Q11 só não foi

selecionada para a criação de VMs pela heurística de expectativas porque o ganho mínimo foi de 50%. Caso não houvesse esta margem, e todas as VMs consideradas benéficas fossem selecionadas, as 6.966 tuplas poderiam ter afetado o resultado final e seria materializada uma VM prejudicial à execução da carga de trabalho.

Ações de sintonia fina com ganhos ou prejuízos pequenos (em relação ao custo total de execução) para a carga de trabalho devem ser considerados resultados inconclusivos e que podem ser provados apenas com a materialização da ação no banco de dados e sua posterior conferência.

6.4.4 O SGBD PostgreSQL e visões materializadas

O Outer-Tuning é capaz de selecionar e criar índices de acordo com a carga de trabalho. Após a criação de um índice, o SGBD PostgreSQL passa a considerá-lo automaticamente ao criar um plano de execução. Caso haja na carga de trabalho uma consulta que se beneficie do índice, o SGBD vai utilizá-lo para responder a consulta e consequentemente tornar a execução da carga de trabalho mais rápida.

Quando uma visão materializada é criada no PostgreSQL, existe um novo desafio no processo de sintonia fina: o SGBD não consegue automaticamente utilizar uma visão materializada para responder uma consulta que não esteja declarando a VM como a fonte dos dados. Mesmo que a consulta seja sintaticamente igual à consulta utilizada para criar a VM, o PostgreSQL não reconhece em seu plano de execução que utilizar a VM terá um melhor desempenho que as tabelas originais.

Ao se criar uma visão materializada, na prática está se criando uma nova instância dos dados ou o armazenamento de novos dados gerados a partir do processamento (agregação, ordenação, funções de cálculo, etc) dos dados originais. A partir do momento que se extrai a informação da tabela e se cria uma visão materializada, tem-se duas fontes de dados distintas. A prova disso, é que caso uma das fontes seja atualizada, a outra não sofrerá alteração.

Dessa forma, para que uma consulta utilize uma visão materializada é necessário a declaração explícita através da modificação da consulta. O gerador de planos de execução do PostgreSQL 9.4 não sabe substituir as referências das tabelas do comando SQL, por uma referência à visão materializada – ao menos até o momento em que os experimentos deste trabalho de pesquisa foram realizados.

6.5 Conclusões

Os resultados aqui apresentados mostram que o Outer-Tuning consegue inferir ações de sintonia fina utilizando a ontologia de sintonia fina. Também foi

mostrado que a extensão da heurística foi realizada com sucesso. As três heurísticas de seleção e criação de visões materializadas foram executadas e seus resultados comparados.

Foi apresentada uma análise dos resultados experimentais, onde o resultado das heurísticas foi comparado e justificado. Também foram comentados os pontos importantes observados durante a realização dos testes. Em particular prejuízos oriundos da utilização de VMs, estatísticas do SGBD desatualizadas e a falta de suporte de VMs pelo PostgreSQL.

No Capítulo 7 apresentamos as conclusões da pesquisa e os trabalhos futuros.

7
Conclusões

Apresentam-se aqui as conclusões sobre as contribuições, algumas limitações e restrições e os trabalhos futuros desta dissertação de mestrado.

7.1
Contribuições

Durante esta pesquisa, foi realizada a extensão da ontologia de sintonia fina, foi criado o projeto de arquitetura e implementação do *framework* Outer-Tuning, o processo de extensão da ontologia e uma avaliação experimental do *framework*.

Foi realizada a extensão da ontologia de domínio, com a inclusão de novos conceitos, atributos e relações envolvidos na tarefa de seleção de visões materializadas. Foram necessárias as inclusões na ontologia de novos conceitos para a representação dos conceitos relativos a visões materializadas e visões materializadas hipotéticas. Também foram adicionados atributos aos conceitos já existentes e relacionamentos que se fizeram necessários.

A extensão da ontologia de tarefas foi realizada com a inclusão de três heurísticas de sintonia fina para seleção e criação de visões materializadas. Uma das heurísticas, a heurística de criação de visões materializadas hipotéticas, tem a função de instanciar na ontologia indivíduos concernentes a visões materializadas hipotéticas. Já as outras duas heurísticas, a heurística de benefício e a heurística de expectativas, são responsáveis por escolher, dentre as visões materializadas hipotéticas, quais deverão gerar visões materializadas reais.

A heurística de expectativas, aqui proposta, também é uma contribuição desta pesquisa. Ela foi desenvolvida com o intuito de ilustrar como o DBA pode incorporar na ontologia sua experiência para criar novas ações de sintonia fina, ou para adequar heurísticas existentes à sua carga de trabalho. Tudo isso a nível de modelo, através da extensão da ontologia.

Como parte inicial do projeto de arquitetura do *framework* Outer-Tuning, foi apresentado um fluxo de execução para ferramentas de seleção de visões materializadas. Esse fluxo foi utilizado para a posterior análise e projeto do Outer-Tuning. Através dele, foram feitas as abstrações necessárias para o desenvolvimento da arquitetura baseada em componentes de software que se apresentou.

Após a definição do fluxo de execução, foi desenvolvido o projeto arquitetural do *framework* Outer-Tuning. Foi realizada uma proposta baseada em componentes com o intuito de compartimentalizar as tarefas, definindo responsabilidades

individuais para cada componente e aumentando o nível de desacoplamento entre eles. Esta arquitetura atende a todos os requisitos funcionais, definidos no início da pesquisa para o *framework*, e foi implementada.

A arquitetura proposta para o *framework* Outer-Tuning foi efetivamente implementada. Um protótipo do *framework* Outer-Tuning foi desenvolvido para a validação da arquitetura e da extensão da ontologia para a tarefa de seleção de visões materializadas.

Por fim, foi realizada uma avaliação experimental da qualidade dos resultados obtidos a partir da execução do Outer-Tuning para execução de sintonia fina utilizando o *benchmark* de referência TPC-H (TPCC15). Através da execução das três heurísticas de sintonia fina, as ações inferidas foram comparadas e avaliadas para a carga de trabalho utilizada.

7.2 Restrições da pesquisa

Descreve-se aqui as restrições no escopo da pesquisa. São limites impostos para que fossem priorizados os objetivos principais estabelecidos inicialmente, as sugestões feitas na defesa de proposta de pesquisa e para a conclusão da dissertação no tempo previsto.

Custo de manutenção e armazenamento das visões materializadas

Existem cargas de trabalho conhecidas como OLTP (*Online Transaction Processing*) e também OLAP (*Online Analytical Processing*). Cargas OLAP fornecem métodos de acessar, visualizar, e analisar um grande volume de dados sob múltiplas perspectivas. Isso pressupõe que os dados não se alteram com frequência. Já cargas OLTP são cargas de trabalho que se encarregam de registrar operações organizacionais como sistemas de vendas, reservas de viagens, entre outras. Para cargas de trabalho assim, a manutenção (alteração / inclusão / exclusão) dos dados é uma tarefa frequente. Isso implica que para cada atualização de uma tabela, as visões materializadas que possuem em sua definição uma referência aos dados alterados, devem ser atualizadas. Existem muitas estratégias para isso, como:

- **Manutenção imediata**: ocorre quando a visão materializada é recalculada imediatamente após pelo menos uma das tabelas que compõe a sua definição terem os dados alterados.
- **Manutenção Periódica**: a tarefa de recálculo da visão materializada é agendada para intervalos de tempos determinados.

- **Manutenção Preguiçosa (*lazy*)**: as atualizações nas tabelas que fazem parte de sua definição são acumuladas até que algum gatilho (*trigger*) dispare a atualização.

O escopo dessa pesquisa se restringiu a seleção e criação do conjunto de visões materializadas dada uma carga de trabalho, mas não considerou o problema de manutenção dos dados das visões materializadas. Tratou-se a carga de dados do tipo OLAP como estática, sem alterações durante as execuções das consultas. Para a utilização das heurísticas apresentadas aqui com cargas de dados OLTP são necessários ajustes no modelo de custo para durante o processo de seleção das VMs que serão persistidas no banco de dados, prever os custos de manutenção das mesmas de acordo com a carga de trabalho esperada.

Apesar de não considerar o custo de manutenção, foi considerado nesta dissertação o custo de armazenamento das VMs como um custo a se minimizar. O Custo de armazenamento foi considerado como um limite superior de espaço disponível para as ações de sintonia fina. Durante a seleção das visões materializadas reais, garantiu-se que a soma total do espaço ocupado em memória secundária não ultrapassasse este limite, que é um parâmetro ajustável do Outer-Tuning e permite que o DBA controle o espaço disponível durante a atividade de sintonia fina.

7.2.1 Consultas aninhadas

Uma consulta SQL é considerada aninhada quando está dentro de outra consulta SQL. Um exemplo é a consulta QX da carga de trabalho do TPC-H utilizado:

```
1  /* TPC-H Small-Quantity-Order Revenue Query (Q17) */
2  select
3          sum(l_extendedprice) / 7.0 as avg_yearly
4  from
5          lineitem, part
6  where
7          p_partkey = l_partkey
8          and p_container = 'MED_BOX'
9          and l_quantity < (
10                         select 0.2 * avg(l_quantity)
11                                 from
12                                         lineitem
13                                 where
14                                         l_partkey = p_partkey
15                         );
```

Nesta consulta, na linha 11, o atributo *l_quantity* possui a restrição de ser menor que o valor retornado pela consulta aninhada. Este é um exemplo relativamente simples de consulta aninhada. Porém pode-se ter um número muito grande de consultas aninhadas uma dentro de outras.

Para a seleção e criação de visões materializadas, as consultas aninhadas exigem um algoritmo de extração e combinação das consultas aninhadas. No exemplo da consulta Q17 do TPC-H pode-se ter duas visões materializadas compostas, uma envolvendo a consulta completa e outra específica para a consulta aninhada. O problema cresce proporcionalmente com a quantidade de consultas aninhadas uma dentro da outras. Alguns autores resolveram o tratamento das consultas aninhadas através de uma modelagem baseada de grafos (Kumar13)(Firmino2011), onde cada possível VM foi considerada um nó em um grafo, e o aninhamento entre duas VMs uma aresta com um peso definido por uma função de custo.

Pela complexidade do problema, pelo tempo disponível para realizar o mestrado e pela prioridade das tarefas, esta pesquisa não tratou de consultas aninhadas. O baixo acoplamento das funções de bibliotecas que reescrevem as consultas SQL e propõem a definição das visões materializadas, permitiu considerar que o tratamento das consultas aninhadas poderia ser realizada como um trabalho futuro através de uma evolução da biblioteca que executa o algoritmo *DefineView* detalhado na Seção 4.2. Desta forma, os esforços foram priorizados para a extensão da ontologia e para o projeto de arquitetura do *framework*.

7.2.2 Índices

A ontologia proposta por Ana Carolina de Almeida (Almeida13) em sua tese de doutorado possui heurísticas para tarefa de seleção, criação e manutenção de índices. Não foi tratado nessa pesquisa a combinação das estratégias de índices e visões materializadas.

A extensão da ontologia para a seleção de visões materializadas criou novos conceitos, atributos e relacionamentos na ontologia de domínio. Alguns atributos de conceitos também foram renomeados para que fosse feita uma generalização de seu uso, onde pudessem ser utilizados para índices e visões. Com isso, as regras SWRL escritas por (Almeida13) para a seleção, criação e manutenção de índices precisariam ser revistas e adequadas aos novos conceitos e à nova disposição dos atributos de conceitos compartilhados entre as duas abordagens. Portanto, o trabalho de adequação das regras SWRL que tratam índices e a combinação das estratégias de índices e visões materializadas foi considerado como trabalhos futuros.

7.3 Trabalhos futuros

Apresentamos aqui possíveis oportunidades de pesquisas e próximos passos na evolução da pesquisa.

Evolução da interface: Existem muitas possibilidades para o avanço na qualidade da interface utilizada pelo DBA para controlar o Outer-Tuning e visualizar as ações de sintonia fina propostas. Uma delas, é a interface realizar de forma automática a comparação entre as ações de sintonia fina. Durante a geração dos resultados experimentais apresentados no Capítulo 6, os dados foram extraídos do Outer-Tuning e colocados no Excel para a geração dos gráficos e os cálculos apresentados. Porém, essa é uma operação braçal que pode ser automatizada e fornecida ao DBA em tempo de execução. Por exemplo: o DBA escolhe N ações de sintonia fina hipotéticas ou reais, e a interface exibe uma comparação entre as suas estatísticas. Na atual implementação, todas as informações necessárias já são disponibilizadas pelo *framework* à sua interface, mas ainda falta uma revisão da forma de apresentar esses dados ao DBA.

Inclusão do custo de manutenção nas regras de seleção de VMs: Uma das restrições dessa pesquisa foi não considerar o custo de manutenção na seleção de VMs. Para que a ferramenta possa atender a outros tipos de cargas de trabalho, como cargas OLTP, é necessário incluir a premissa de que os dados poderão sofrer alterações. Isso inclui definir uma política de atualização para as VMs e alterar as heurísticas da ontologia para calcular o custo de manutenção e executar a atualização das VMs quando necessário.

Criar um analisador sintático (*parser*) para tradução de heurísticas em regras SWRL/DL: O Outer-Tuning trouxe a capacidade de definir regras de sintonia fina através da extensão da sua ontologia. Não há a necessidade de manipular código-fonte para incluir ou alterar uma nova heurística. Porém, as heurísticas são expressas em linguagens declarativas como SWRL ou DL. Estas linguagens podem ter um nível de complexidade elevado para que um DBA inexperiente possa alterá-las com eficiência. Uma solução seria criar um analisador sintático capaz de traduzir heurísticas de sintonia fina em regras SWRL ou DL.

Gerar justificativas de ações de sintonia fina automáticas, sem modelos e em linguagem natural: Atualmente as justificativas de sintonia fina são geradas através de templates pré-formatados. Uma continuação possível da pesquisa, seja criar justificativas semânticas automáticas e em linguagem natural para apoiar o DBA na decisão de quais ações de sintonia fina devem ser tomadas.

Consultas aninhadas: Não foi considerado no escopo desta pesquisa o tratamento das consultas SQL aninhadas. É um trabalho futuro o estudo das melhores estratégias de tratamento das consultas aninhadas e como pode ser

realizada a instanciação entre as possíveis visões materializadas derivadas de uma única consulta.

Adequação das regras de índices: A ontologia inicialmente propunha a seleção, criação e manutenção de índices. Após a extensão para a seleção e criação de visões materializadas, as regras de índice precisam ser revistas e adequadas à nova disposição da ontologia de domínio. Também faz parte deste trabalho futuro a comparação, e a combinação das estratégias de índices e visões materializadas. Esta pesquisa mostrou que heurísticas para um mesmo tipo de ação de sintonia fina podem coexistir e trabalhar simultaneamente para gerar ações de sintonia fina úteis para a escolha do DBA. Logo, nada impede a comparação entre heurísticas diferentes e a combinação entre elas.

Referências Bibliográficas

[Almeida13] ALMEIDA, A. C. B.. **Framework para apoiar a sintonia fina de banco de dados**. PhD thesis, Pontifícia Universidade Católica do Rio de Janeiro - PUC-RIO, 2013. (document), 1.1, 1.2, 1.3, 2.11, 3.1, 3.3, 3.4, 4, 4.1, 4.2.1, 4.4, 4.2.1, 5.1, 7.2.2

[Aouiche06] AOUICHE, K.; JOUVE, P.-E. ; DARMONT, J.. **Clustering-based materialized view selection in data warehouses**. In: PROCEEDINGS OF THE 10TH EAST EUROPEAN CONFERENCE ON ADVANCES IN DATABASES AND INFORMATION SYSTEMS, ADBIS, p. 81–95, 2006. 3.2

[Azevedo09] AZEVEDO, L. G.; BAIÃO, F. A. ; SANTORO, F.. **Identificação de Serviços a partir da Modelagem de Processos de Negócio**. Anais do V Simpósio Brasileiro de Sistemas de Informação - SBSI, p. 133–144, 2009. 5.2.2

[Breitman07] BREITMAN, K.; CASANOVA, M. A. ; TRUSZKOWSKI, W.. **Semantic Web: Concepts, Technologies and Applications**. NASA Monographs in Systems and Software Engineering. Springer, 2007. 2.3

[Brown96] BROWN, A. W.; WALLNAU, K. C.. **Engineering of Component Based Systems**. In: COMPONENT-BASED SOFTWARE ENGINEERING, p. 7–15, 1996. 5.2.2

[Bruno2012] BRUNO, N.. **Automated Physical Database Design and Tuning**. CRC Press, 2012. 2.7, 2.8

[Carvalho11] CARVALHO, A. W.. **Criação Automática de Visões Materializadas em SGBDs Relacionais**. Master's thesis, Pontifícia Universidade Católica do Rio de Janeiro - PUC-RIO, 2011. 2.10, 2.11, 3.1, 4.2.1, 4.2.1, 4.2.2, 4.2.2, 4.2.2

[Chan01] CHAN, G.; LI, Q. ; FENG, L.. **Optimized design of materialized views in a real-life data warehousing environment**. International Journal of Information Technology, 7(1):30–54, 2001. 3.2

[Chirkova11] CHIRKOVA, R.; YANG, J.. **Materialized views**. Foundations and Trends in Databases, 4(4):295–405, 2012. 2.10, 2.10.1, 2.10.2

[Costa2003] COSTA, R. L. D. C.; LIFSCHITZ, S. ; SALLES, M. A. V.. **Index Self-tuning with Agent-based Databases**. In: PROCEEDINGS OF THE LATIN-AMERICAN CONFERENCE ON INFORMATICS (CLEI), volumen 6, p. 1–22, 2003. 2.11

[Costa05] COSTA, R. D. C.; LIFSCHITZ, S.; DE NORONHA, M. ; VAZ SALLES, M.. **Implementation of an agent architecture for automated index tuning**. In: PROCEEDINGS OF DATA ENGINEERING WORKSHOPS, 2005. 21ST INTERNATIONAL CONFERENCE ON, p. 1215–1215, 2005. 3.1

[Date2004] DATE, C. J.. **Introdução a sistemas de bancos de dados**. Campus, 2004. 2.6

[Derakhshan06] DERAKHSHAN, R.; DEHNE, F.. **Simulated Annealing for Materialized View Selection in Data Warehousing Environment.** In: PROCEEDINGS OF IASTED INTL. CONF. ON DATABASE AND APPLICATIONS, p. 89–94, 2006. 3.2

[Derakshan08] DERAKHSHAN, R.; STANTIC, B.; KORN, O. ; DEHNE, F.. **Parallel simulated annealing for materialized view selection in data warehousing environments**. volumen 5022 de **Lecture Notes in Computer Science**, p. 121–132. 2008. 3.2

[Fayad99] FAYAD, M.; SCHMIDT, D. C. ; JOHNSON, R. E.. **Building application frameworks: object-oriented foundations of framework design**. Wiley computer publishing. 1999. 2.5

[Firmino2011] FIRMINO, A.; MATEUS, R.. **A novel method for selecting and materializing views based on OLAP signatures and grasp**. Journal of Information and Data Management, 2(3):479–494, 2011. 7.2.1

[Friedman14] FRIEDMAN-HILL, E.. **Jess, the Rule Engine for the JavaTM Platform, http://www.jessrules.com/**, 2014. 5.2.2

[Fuks11] FUKS, H.; PIMENTEL, M.. **Sistemas Colaborativos**. Rio de Janeiro, campos edition, 2011. 2.4

[Gao10] SONG, X.; GAO, L.. **An ant colony based algorithm for optimal selection of materialized view**. In: PROCEEDINGS OF INTELLIGENT COMPUTING AND INTEGRATED SYSTEMS (ICISS), p. 534–536, 2010. 3.2

[Gruber09] GRUBER, T.. **Ontologies**. In: PROCEEDINGS OF ENCYCLOPEDIA OF DATABASE SYSTEMS, p. 1959–1959. Springer, 2009. 2.1

[Guarino98] GUARINO, N.. **Formal ontology and information systems**. In: PROCEEDINGS OF ONTOLOGY IN INFORMATION SYSTEMS, p. 3–15, 1998. 2.1

[Hose09] HOSE, K.; KLAN, D. ; SATTLER, K. U.. **Online tuning of aggregation tables for OLAP**. In: PROCEEDINGS OF INTERNATIONAL CONFERENCE ON DATA ENGINEERING, p. 1679–1686, 2009. 4.2.2, 4.2.2

[ISO2015] ISO. **SQL:2011**, 2015. 4.2.1

[Khosla04] KHOSLA, R.; ICHALKARANJE, N. ; JAIN, L. C.. **Design of Intelligent Multi-Agent Systems: Human-Centredness, Architectures, Learning and Adaptation**. Studies in Fuzziness and Soft Computing. Springer, 2004. 5.2.2

[Kumar09] VIJAY KUMAR, T.; GHOSHAL, A.. **A reduced lattice greedy algorithm for selecting materialized views**. In: INFORMATION SYSTEMS, TECHNOLOGY AND MANAGEMENT, volumen 31 de **Communications in Computer and Information Science**, p. 6–18. 2009. 3.2

[Kumar12] VIJAY KUMAR, T.; KUMAR, S.. **Materialized view selection using genetic algorithm**. In: CONTEMPORARY COMPUTING, volumen 306 de **Communications in Computer and Information Science**, p. 225–237. 2012. 3.2

[Kumar13] VIJAY KUMAR, T.; KUMAR, S.. **Materialized view selection using iterative improvement**. In: ADVANCES IN COMPUTING AND INFORMATION TECHNOLOGY, volumen 178 de **Advances in Intelligent Systems and Computing**, p. 205–213. 2013. 3.2, 7.2.1

[Lawrence06] LAWRENCE, M.. **Multiobjective genetic algorithms for materialized view selection in OLAP data warehouses**. In: PROCEEDINGS OF GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE - GECCO. ACM Press, 2006. 3.2

[Li10a] LI, X.; QIAN, X.; JIANG, J. ; WANG, Z.. **Shuffled frog leaping algorithm for materialized views selection**. In: PROCEEDINGS OF

2010 SECOND INTERNATIONAL WORKSHOP ON EDUCATION TECHNOLOGY AND COMPUTER SCIENCE (ETCS), volumen 3, p. 7–10, 2010. 3.2

[Milanes15] MILANÉS, A. Y.; LIFSCHITZ, S.. **Design and Implementation of a Global Self-tuning Architecture**. Anais do Simpósio Brasileiro de Banco de Dados - SBBD, p. 70–84, 2005. 3.1

[Monteiro08] MONTEIRO, J. M.; BRAYNER, A. ; LIFSCHITZ, S.. **Estado da Arte em Auto-Sintonia do Projeto Físico de BD.** MCC - Monografias em Ciência da Computação - PUC-RIO, 2008. 1.1, 2.8, 2.9, 2.10, 2.11, 3.1, 4.2.2, 4.2.2

[Morelli06] MORELLI, E. M. T.. **Recriação Automática de Índices em um SGBD Relacional**. Master's thesis, Pontifícia Universidade Católica do Rio de Janeiro - PUC-RIO, 2006. 2.11, 3.1, 4.2.2, 4.2.2

[Noy04] NOY, N. F.; DOAN, A. ; HALEVY, A. Y.. **Semantic integration: A survey of ontology-based approaches**. SIGMOD, 33(4):65–70, 2004. 2.1

[Oracle2014] ORACLE. **Java**, 2014. 5.2.2

[Phuboon07] PHUBOON-OB, J.; AUEPANWIRIYAKUL, R.. **Selecting Materialized Views Using Two-Phase Optimization with Multiple View Processing Plan**. Intl. Journal of Computer & Information Science & Engine, p. 166–171, 2007. 3.2

[Ramarkrishnan08] RAMARKRISHNAN, R.; GEHRKE, J.. **Sistemas de gerênciamento de banco de dados**. McGraw Hill, 2008. 2.6, 2.7, 2.8, 2.10, 2.10.1

[Salles04] SALLES, M. A. V.. **Criação autônoma de índices em bancos de dados**. Master's thesis, Pontifícia Universidade Católica do Rio de Janeiro, 2004. 2.7, 2.11, 3.1, 4.2.2, 4.2.2

[Shasha02] SHASHA, D.; BONNET, P.. **Database Tuning: Principles, Experiments, and Troubleshooting Techniques**. Elsevier Science, 2002. 2.7

[Silberschatz2012] SILBERSCHATZ, A.; KORTH, H. F. ; SUDARSHAN, S.. **Sistema de banco de dados**. CAMPUS - RJ, 2012. 2.10

[Sommerville2011] SOMMERVILLE, I.. **Software Engineering**. Pearson Education, 2011. 5.2.2

[Sun09] SUN, X.; WANG, Z.. **An Efficient Materialized Views Selection Algorithm Based on PSO**. In: PROCEEDINGS OF WORKSHOP ON INTELLIGENT SYSTEMS AND APPLICATIONS, p. 1–4. IEEE, 2009. 3.2

[TPCC15] PERFORMANCE, C. T. P.. **TPC-C.http://www.tpc.org/tpcc/**, 2015. 1.2, 3.1, 7.1

[Talebian09] TALEBIAN, S.; ABDUL KAREEM, S.. **Using genetic algorithm to select materialized views subject to dual constraints**. In: PROCEEDINGS OF INTERNATIONAL CONFERENCE ON SIGNAL PROCESSING SYSTEMS, p. 633–638, 2009. 3.2

[Talebian10] TALEBIAN, S. H.; KAREEM, S. A.. **A lexicographic ordering genetic algorithm for solving multi-objective view selection problem**. In: PROCEEDINGS OF THE SECOND INTERNATIONAL CONFERENCE ON COMPUTER RESEARCH AND DEVELOPMENT, ICCRD, p. 110–115. IEEE Computer Society, 2010. 3.2

[W3C14] W3C. **SWRL Language, http://www.w3.org/Submission/SWRL**, 2014. 2.2, 2.3

[Yhang10] YUHANG, Z.; QI, L. ; WEI, Y.. **Materialized view selection algorithm**. In: PROCEEDINGS OF COMPUTATIONAL INTELLIGENCE AND NATURAL COMPUTING PROCEEDINGS (CINC), 2010 SECOND INTERNATIONAL CONFERENCE ON, volumen 1, p. 68–71, 2010. 3.2

A
Consultas TPC-H

A.1
Consulta original TPCH-H - Q01

```
1  /* TPC-H/TPC-R Pricing Summary Report Query (Q1) */
2
3  select
4          l_returnflag,
5          l_linestatus,
6          sum(l_quantity) as sumqty,
7          sum(l_extendedprice) as sumbaseprice,
8          sum(l_extendedprice * (1 - l_discount))      as
               sumdiscprice,
9          sum(l_extendedprice * (1 - l_discount) *      (1 + l_tax))
               as sumcharge,
10         avg(l_quantity) as avgqty,
11         avg(l_extendedprice) as avgprice,
12         avg(l_discount) as avgdisc,
13         count(*) as countorder
14 from
15         lineitem l
16 where
17         l_shipdate <= date '1998-12-01' - interval '87 days'
18 group by
19         l_returnflag,
20         l_linestatus
21 order by
22         l_returnflag,
23         l_linestatus;
```

A.2
Consulta TPCH-H - Q01 reescrita para VM

```
1  /* TPC-H/TPC-R Pricing Summary Report Query (Q1) */
2  select
3          l_returnflag,
4          l_linestatus,
5          sum(sumqty) as sumqty,
6          sum(sumbaseprice) as sumbaseprice,
7          sum(sumdiscprice) as sumdiscprice,
8          sum(sumcharge) as sumcharge,
9          avg(avgqty) as avgqty,
10         avg(avgprice) as avgprice,
11         avg(avgdisc) as avgdisc,
12         count(*) as countorder
13 from
14         public.v_ot_workload_1
15 where
16         l_shipdate <= date '1998-12-01' - interval '87 days'
17 group by
18         l_returnflag,
19         l_linestatus
20 order by
21         l_returnflag,
22         l_linestatus;
```

A.3
Consulta TPCH-H - Q03

```
1  /* TPC-H/TPC-R Shipping Priority Query (Q3) */
2  select
3          l_orderkey,
4          sum(l_extendedprice * (1 - l_discount)) as revenue,
5          o_orderdate, o_shippriority
6  from
7          customer, orders, lineitem
8  where
9          c_mktsegment = 'BUILDING'
10         and c_custkey = o_custkey
11         and l_orderkey = o_orderkey
12         and o_orderdate < date '1995-03-15'
13         and l_shipdate > date '1995-03-15'
14 group by
15         l_orderkey,
16         o_orderdate,
17         o_shippriority
18 order by
19         revenue desc,
20         o_orderdate;
```

A.4
Consulta TPCH-H - Q03 reescrita para VM

```
1  /* TPC-H/TPC-R Shipping Priority Query (Q3) */
2  select
3          l_orderkey,
4          sum(revenue) as revenue,
5          o_orderdate, o_shippriority
6  from
7          public.v_ot_workload_5
8  where
9          c_mktsegment = 'BUILDING'
10         and o_orderdate < date '1995-03-15'
11         and l_shipdate > date '1995-03-15'
12 group by
13         l_orderkey,
14         o_orderdate,
15         o_shippriority
16 order by
17         revenue desc,
18         o_orderdate;
```

A.5
Consulta TPCH-H - Q04

```
1  /* TPC-H/TPC-R Order Priority Checking Query (Q4) */
2  select
3          o_orderpriority,
4          count(*) as order_count
5  from
6          orders,
7          lineitem
8  where
9          o_orderdate >= date '1993-07-01'
10         and o_orderdate < date '1993-07-01' + interval '3' month
11         and l_orderkey = o_orderkey
12         and l_commitdate < l_receiptdate
13 group by
14         o_orderpriority
15 order by
16         o_orderpriority;
```

A.6
Consulta TPCH-H - Q04 reescrita para VM

```
1  /* TPC-H/TPC-R Order Priority Checking Query (Q4) */
2  select
3          o_orderpriority,
4          count(*) as order_count
5  from
6          public.v_ot_workload_8
7  where
8          o_orderdate >= date '1993-07-01'
9          and o_orderdate < date '1993-07-01' + interval '3' month
10 group by
11         o_orderpriority
12 order by
13         o_orderpriority;
```

A.7
Consulta TPCH-H - Q05

```
1  /* TPC-H/TPC-R Local Supplier Volume Query (Q5) */
2  select
3          n_name,
4          sum(l_extendedprice * (1 - l_discount)) as revenue
5  from
6          customer,        orders,
7          lineitem,        supplier,
8          nation, region
9  where
10         c_custkey = o_custkey
11         and l_orderkey = o_orderkey
12         and l_suppkey = s_suppkey
13         and c_nationkey = s_nationkey
14         and s_nationkey = n_nationkey
15         and n_regionkey = r_regionkey
16         and r_name = 'ASIA'
17         and o_orderdate >= date '1994-01-01'
18         and o_orderdate < date '1994-01-01' + interval '1' year
19 group by
20         n_name
21 order by
22         revenue desc;
```

A.8
Consulta TPCH-H - Q05 reescrita para VM

```
1  /* TPC-H/TPC-R Local Supplier Volume Query (Q5) */
2  select
3          n_name,
4          sum(revenue) as revenue
5  from
6          public.v_ot_workload_14
7  where
8          r_name = 'ASIA'
9          and o_orderdate >= date '1994-01-01'
10         and o_orderdate < date '1994-01-01' + interval '1' year
11 group by
12         n_name;
```

A.9
Consulta TPCH-H - Q06

```
1  /* TPC-H/TPC-R Forecasting Revenue Change Query (Q6) */
2  select
3          sum(l_extendedprice * l_discount) as revenue
4  from
5          lineitem
6  where
7          l_shipdate >= date '1994-01-01'
8          and l_shipdate < date '1994-01-01' + interval '1' year
9          and l_discount between 0.06 - 0.01 and 0.06 + 0.01
10         and l_quantity < 24;
```

A.10
Consulta TPCH-H - Q06 reescrita para VM

```
1  /* TPC-H/TPC-R Forecasting Revenue Change Query (Q6) */
2  select
3          sum(revenue) as revenue
4  from
5          public.v_ot_workload_13
6  where
7          l_shipdate >= date '1994-01-01'
8          and l_shipdate < date '1994-01-01' + interval '1' year
9          and l_discount between 0.06 - 0.01 and 0.06 + 0.01
10         and l_quantity < 24;
```

A.11
Consulta TPCH-H - Q07

```
1  /* TPC-H/TPC-R Volume Shipping Query (Q7) */
2  select
3          n_name as supp_nation,
4          n_name as cust_nation,
5          date_part('year', l_shipdate) as l_year,
6          sum(l_extendedprice * (1 - l_discount)) as volume
7  from
8          supplier, lineitem,      orders, customer, nation
9  where
10         s_suppkey = l_suppkey
11         and o_orderkey = l_orderkey
12         and c_custkey = o_custkey
13         and s_nationkey = n_nationkey
14         and (n_name = 'FRANCE' or n_name = 'GERMANY')
15         and l_shipdate between date '1991-01-01' and date '
               1998-12-31'
16 group by
17         supp_nation, cust_nation, l_year
18 order by
19         supp_nation, cust_nation,
20         l_year;
```

A.12
Consulta TPCH-H - Q07 reescrita para VM

```
1  /* TPC-H/TPC-R Volume Shipping Query (Q7) */
2  select
3          supp_nation,
4          cust_nation,
5          l_year,
6          sum(volume) as volume
7  from
8          public.v_ot_workload_10
9  where
10         (supp_nation = 'FRANCE' or supp_nation = 'GERMANY')
11         and l_shipdate between date '1991-01-01' and date '
               1998-12-31'
12 group by
13         supp_nation, cust_nation, l_year
14 order by
15         supp_nation, cust_nation,
16         l_year;
```

A.13
Consulta TPCH-H - Q08

```
1  /* TPC-H/TPC-R National Market Share Query (Q8) */
2  select
3          sum(case
4                  when n_name = 'BRAZIL' then (l_extendedprice * (1
                       - l_discount))
5                  else 0
6          end) / sum(l_extendedprice * (1 - l_discount)) as
               mkt_share,
7          date_part('year', o_orderdate) as o_year
8  from
9          part,
10         supplier,
11         lineitem,
12         orders,
13         customer,
14         nation,
15         region
16 where
17         p_partkey = l_partkey
18         and s_suppkey = l_suppkey
19         and l_orderkey = o_orderkey
20         and o_custkey = c_custkey
21         and c_nationkey = n_nationkey
22         and n_regionkey = r_regionkey
23         and r_name = 'AMERICA'
24         and o_orderdate between date '1995-01-01' and date '
               1996-12-31'
25         and p_type = 'ECONOMY ANODIZED STEEL'
26 group by o_year
27 order by o_year;
```

A.14
Consulta TPCH-H - Q09

```
1  /* TPC-H/TPC-R Product Type Profit Measure Query (Q9) */
2  select
3          n_name as nation,
4          date_part('year', o_orderdate) as o_year,
5          sum(l_extendedprice * (1 - l_discount) - ps_supplycost *
               l_quantity) as sum_profit
6  from
7          part, supplier, lineitem,
8      partsupp, orders, nation
9  where
10         s_suppkey = l_suppkey
11         and ps_suppkey = l_suppkey
12         and ps_partkey = l_partkey
13         and p_partkey = l_partkey
14         and o_orderkey = l_orderkey
15         and s_nationkey = n_nationkey
16         and p_name like '%green%'
17 group by
18         nation, o_year
19 order by
20         nation, o_year desc;
```

A.15
Consulta TPCH-H - Q09 reescrita para VM

```
1  /* TPC-H/TPC-R Product Type Profit Measure Query (Q9) */
2  select
3          nation,
4          o_year,
5          sum(sum_profit) as sum_profit
6  from
7          public.v_ot_workload_6
8  where
9          p_name like '%green%'
10 group by
11         nation, o_year
12 order by
13         nation, o_year desc;
```

A.16
Consulta TPCH-H - Q10

```
1  /* TPC–H/TPC–R Returned Item Reporting Query (Q10) */
2  select
3          c_custkey,
4          c_name,
5          sum(l_extendedprice * (1 - l_discount)) as revenue,
6          c_acctbal,
7          n_name,
8          c_address,
9          c_phone,
10         c_comment
11 from
12         customer,
13         orders,
14         lineitem,
15         nation
16 where
17         c_custkey = o_custkey
18         and l_orderkey = o_orderkey
19         and o_orderdate >= date '1993-10-01'
20         and o_orderdate < date '1993-10-01' + interval '3' month
21         and l_returnflag = 'R'
22         and c_nationkey = n_nationkey
23 group by
24         c_custkey,
25         c_name,
26         c_acctbal,
27         c_phone,
28         n_name,
29         c_address,
30         c_comment
31 order by
32         revenue desc
33 limit 20;
```

A.17
Consulta TPCH-H - Q11

```
1  /* TPC–H/TPC–R Important Stock Identification Query (Q11) */
2  select
3          ps_partkey,
4          sum(ps_supplycost * ps_availqty) as value
5  from
6          partsupp,
7          supplier,
8          nation
9  where
10         s_nationkey = n_nationkey
11         and n_nationkey = 24
12 group by
13         ps_partkey
14 having
15         sum(ps_supplycost * ps_availqty) > sum(ps_supplycost *
               ps_availqty) * 0.0001000000
16 order by
17         value desc;
```

A.18
Consulta TPCH-H - Q12

```
1  /* TPC-H/TPC-R Shipping Modes and Order Priority Query (Q12) */
2  select
3          l_shipmode,
4          sum(case
5                  when o_orderpriority = '1-URGENT'
6                  or o_orderpriority = '2-HIGH'
7                          then 1
8                  else 0
9          end) as high_line_count,
10         sum(case
11                 when o_orderpriority <> '1-URGENT'
12                 and o_orderpriority <> '2-HIGH'
13         then 1
14                 else 0
15         end) as low_line_count
16 from
17         orders, lineitem
18 where
19         o_orderkey = l_orderkey
20         and l_shipmode in ('MAIL', 'SHIP')
21         and l_commitdate < l_receiptdate
22         and l_shipdate < l_commitdate
23         and l_receiptdate < date '1994-01-01' + interval '1' year
24 group by l_shipmode
25 order by l_shipmode;
```

A.19
Consulta TPCH-H - Q12 reescrita para VM

```
1  /* TPC-H/TPC-R Shipping Modes and Order Priority Query (Q12) */
2  select
3          l_shipmode,
4          sum(high_line_count) as high_line_count,
5          sum(low_line_count) as low_line_count
6  from
7          public.v_ot_workload_4
8  where
9          l_shipmode in ('MAIL', 'SHIP')
10         and l_receiptdate < date '1994-01-01' + interval '1' year
11 group by l_shipmode
12 order by l_shipmode;
```

A.20
Consulta TPCH-H - Q14

```
1  /* TPC-H/TPC-R Promotion Effect Query (Q14) */
2  select
3          100.00 * sum(case
4                  when p_type like 'PROMO%'
5                          then l_extendedprice * (1 - l_discount)
6                  else 0
7          end) / sum(l_extendedprice * (1 - l_discount)) as
               promo_revenue
8  from
9          lineitem,
10         part
11 where
12         l_partkey = p_partkey
13         and l_shipdate >= date '1995-09-01'
14         and l_shipdate < date '1995-09-01' + interval '3' month;
```

A.21
Consulta TPCH-H - Q14 reescrita para VM

```
1  /* TPC-H/TPC-R Promotion Effect Query (Q14) */
2  select
3          sum(promo_revenue) as promo_revenue
4  from
5          public.v_ot_workload_7
6  where
7          l_shipdate >= date '1995-09-01'
8          and l_shipdate < date '1995-09-01' + interval '3' month;
```

A.22
Consulta TPCH-H - Q16

```
1  /* TPC-H/TPC-R Parts/Supplier Relationship Query (Q16) */
2  select
3          p_brand,
4          p_type,
5          p_size,
6          count(distinct ps_suppkey) as supplier_cnt
7  from
8          partsupp,
9          supplier,
10         part
11 where
12         p_partkey = ps_partkey
13         and p_brand <> 'Brand#45'
14         and p_type not like 'MEDIUM_POLISHED%'
15         and p_size in (49, 14, 23, 45, 19, 3, 36, 9)
16         and ps_suppkey <> s_suppkey
17         and      s_comment like '%Customer%Complaints%'
18 group by
19         p_brand,
20         p_type,
21         p_size
22 order by
23         supplier_cnt desc,
24         p_brand,
25         p_type,
26         p_size;
```

A.23
Consulta TPCH-H - Q19

```
1  /* TPC-H/TPC-R Discounted Revenue Query (Q19) */
2  select
3          sum(l_extendedprice* (1 - l_discount)) as revenue
4  from
5          lineitem,
6          part
7  where
8          (
9                  p_partkey = l_partkey
10                 and p_brand = 'Brand#12'
11                 and p_container in ('SM_CASE', 'SM_BOX', 'SM_PACK'
                       , 'SM_PKG')
12                 and l_quantity >= 1 and l_quantity <= 1 + 10
13                 and p_size between 1 and 5
14                 and l_shipmode in ('AIR', 'AIR_REG')
15                 and l_shipinstruct = 'DELIVER_IN_PERSON'
16         )
17         or
18         (
19                 p_partkey = l_partkey
20                 and p_brand = 'Brand#23'
21                 and p_container in ('MED_BAG', 'MED_BOX', 'MED_PKG
                       ', 'MED_PACK')
22                 and l_quantity >= 10 and l_quantity <= 10 + 10
23                 and p_size between 1 and 10
24                 and l_shipmode in ('AIR', 'AIR_REG')
25                 and l_shipinstruct = 'DELIVER_IN_PERSON'
26         )
27         or
28         (
29                 p_partkey = l_partkey
30                 and p_brand = 'Brand#34'
31                 and p_container in ('LG_CASE', 'LG_BOX', 'LG_PACK'
                       , 'LG_PKG')
32                 and l_quantity >= 20 and l_quantity <= 20 + 10
33                 and p_size between 1 and 15
34                 and l_shipmode in ('AIR', 'AIR_REG')
35                 and l_shipinstruct = 'DELIVER_IN_PERSON'
36         );
```