UNIVERSIDADE FEDERAL DO CEARÁ

ARLINO HENRIQUE MAGALHÃES DE ARAÚJO

ABORDAGENS NÃO-INTRUSIVAS PARA SINTONIA DE INSTRUÇÕES SQL

FORTALEZA, CEARÁ

2012

ARLINO HENRIQUE MAGALHÃES DE ARAÚJO

ABORDAGENS NÃO-INTRUSIVAS PARA SINTONIA DE INSTRUÇÕES SQL

Dissertação apresentada no Programa de Mestrado e Doutorado em Ciência da Computação da Universidade Federal do Ceará, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Banco de Dados

Orientador: José Maria da Silva Monteiro Filho

Co-Orientador: José Antônio Fernandes de Macêdo

FORTALEZA, CEARÁ

2012

A000z Araújo, Arlino Henrique Magalhães de.
Abordagens Não-Intrusivas para Sintonia de Instruções SQL / Arlino Henrique Magalhães de Araújo. – Fortaleza, 2012.
141p.;il.
Orientador: Prof. Dr. José Maria da Silva Monteiro Filho
Co-Orientador: José Antônio Fernandes de Macêdo
Dissertação (Programa de Mestrado e Doutorado em Ciência da Computação) - Universidade Federal do Ceará, Centro de Ciências Científicas.
1. Banco de Dados 2. Sintonia de Banco de Dados 3. Sintonia de Instruções SQL I. Universidade Federal do Ceará, Centro de Ciências Científicas.
~~CDD:000.0~~

ARLINO HENRIQUE MAGALHÃES DE ARAÚJO

ABORDAGENS NÃO-INTRUSIVAS PARA SINTONIA DE INSTRUÇÕES SQL

Dissertação apresentada no Programa de Mestrado e Doutorado em Ciência da Computação da Universidade Federal do Ceará, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Aprovada em: __/__/____

BANCA EXAMINADORA

Prof. Dr. José Maria da Silva Monteiro Filho
Universidade Federal do Ceará - UFC
Orientador

Prof. Dr. José Antônio Fernandes de Macêdo
Universidade Federal do Ceará - UFC
Co-orientador

Prof. Javam de Castro Machado
Universidade Federal do Ceará - UFC

Prof. Ângelo Roncalli Alencar Brayner
Universidade de Fortaleza - UNIFOR

AGRADECIMENTOS

Agradeço em primeiro lugar a Deus que iluminou o meu caminho durante esta caminhada.

Aos meus pais, irmãs, minha namorada, meu filho e a toda minha família que, com muito carinho e apoio, não mediram esforços para que eu chegasse até esta etapa de minha vida.

Ao professor José Maria, pela paciência na orientação e incentivo que tornaram possível a conclusão desta dissertação.

A todos os professores do curso, que foram tão importantes na minha vida acadêmica e no desenvolvimento desta dissertação.

Aos amigos e colegas, pelo incentivo e pelo apoio constantes.

O essencial, com efeito, na educação, não é a doutrina ensinada, é o despertar.
Ernest Renan

RESUMO

Sistemas Gerenciadores de Bancos de Dados (SGBDs) permitem especificar comandos, por meio de linguagens declarativas de alto nível, como SQL (Structured Query Language), por exemplo, com a finalidade de executar diferentes operações sobre os dados armazenados (consultas, atualizações, inserções e remoções). O otimizador de consultas é o módulo do SGBD responsável por escolher um plano de execução eficiente para cada comando SQL a ser executado. Para este propósito, os otimizadores procuram, em um grande espaço de busca, o plano de execução que proporcione o menor tempo de resposta. Dois comandos SQL são considerados equivalentes se retornarem os mesmos resultados. Uma vez que a sintaxe de um comando SQL influencia a escolha do plano de execução, o otimizador pode produzir planos de execução distintos para comandos SQL equivalentes. Consequentemente, comandos SQL equivalentes podem apresentar tempos de resposta diferentes. Este fato decorre das operações utilizadas em cada comando SQL (ordenações, agregações, remoção da valores duplicados, utilização de tabelas temporárias, subconsultas, dentre outras). Neste cenário, mesmo usando métodos de acesso e estratégias de avaliação suportadas pelo SGBD, às vezes, os otimizadores não conseguem produzir planos ótimos. Nestes casos, deve ser realizado o ajuste (ou sintonia) do comando SQL. Para isso, em geral, duas estratégias são frequentemente utilizadas: (a) reescrever o comando SQL; e (b) aplicar *Query Hinting*. A técnica de reescrita consiste em escrever um novo comando SQL, equivalente ao comando SQL original, mas que apresente um tempo de resposta menor. Entretanto, o processo de reescrita de comandos SQL é complexo e requer bastante conhecimento em diferentes áreas, tais como: (i) otimização de consultas, execução de operadores de planos de consultas, configuração de parâmetros e outros aspectos internos dos bancos de dados; (ii) identificação de índices necessários e outras estruturas de acesso; (iii) manutenção de estatísticas sobre os dados; e (iv) características dos sistemas de armazenamento de dados. Este trabalho propõe duas abordagens distintas para suportar a reescrita de comandos SQL em bancos de dados relacionais: uma abordagem assistida e outra automática. As duas abordagens propostas utilizam um conjunto de heurísticas para realizar a reescrita dos comandos SQL. As heurísticas são constituídas de regras que visam identificar oportunidades de sintonia nos comandos SQL. Com o objetivo de avaliar a eficiência das abordagens propostas uma avaliação experimental foi realizada. Os experimentos foram conduzidos em três diferentes cenários: i) com o *benchmark* TPC-H, ii) com a base de dados do TPC-H e uma carga de trabalho sintética e iii) com a base de dados do sistema SIG e uma carga de trabalho sintética. Para cada cenário, três SGBDs foram avaliados: PostgreSQL, Oracle e SQL Server. Os resultados dos testes realizados mostram que tanto a abordagem assistida quanto a automática proporcionaram ganhos de desempenho, reduzindo o tempo de resposta das cargas de trabalho avaliadas.

Palavras-Chave: Banco de Dados, Sintonia de Banco de Dados, Sintonia de Instruções SQL

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação e Caracterização do Problema

Sistemas Gerenciadores de Bancos de Dados (SGBDs) permitem especificar comandos, por meio de linguagens declarativas de alto nível, como SQL (*Structured Query Language*), por exemplo, com a finalidade de executar diferentes operações sobre os dados armazenados (consultas, atualizações, inserções e remoções). O otimizador de consultas é o módulo do SGBD responsável por escolher um plano de execução eficiente para cada comando SQL a ser executado. Para este propósito, os otimizadores baseados em custos procuram, em um espaço de busca, o plano de execução que proporcione o menor tempo de resposta. Para realizar esta tarefa para uma determinada consulta, os otimizadores utilizam um modelo de custo, o qual possibilita estimar os recursos necessários e o custo da execução de cada um dos planos analisados, possibilitando assim comparar o custo das várias alternativas de planos para, possivelmente selecionar o plano de execução ótimo, ou seja, que apresente o menor custo de execução.

Os modelos de custo utilizados pelos otimizadores de consulta atuais utilizam algoritmos bastante complexos para encontrar um plano de execução que tenha um custo razoavelmente próximo do custo mínimo possível; e dependem de fatores que podem torná-los imprecisos, tais como:

(a) estimativas de cardinalidade imprecisas, que são inferidas através de informações estatíticas;

(b) constantes de calibração, usadas na estimativa de determinados custos, os quais não podem ser calculados de forma exata ou que dependem de aspectos específicos do *hardware* utilizado, como, por exemplo, o custo de I/O em um disco rígido específico;

(c) fórmulas de cálculo de determinados custos podem não capturar todos os detalhes dos algoritmos utilizados nos planos de execução;

(d) parâmetros de configuração de sistemas de banco de dados podem influenciar os custos de execução das consultas, como, por exemplo, tamanho do *buffer* de memória; e

(e) mesmo que fosse possível determinar com exatidão o custo de uma determinada consulta de maneira isolada, na prática, as consultas executam de forma concorrente e a execução de uma consulta pode influenciar o tempo de execução das demais devido a fatores como o compartilhamento de *buffer* e das estruturas de índice e bloqueios excessivos (BRUNO; CHAUDHURI; RAMAMURTHY, 2009).

Como consequência, muitas vezes, os otimizadores não conseguem produzir planos ótimos, mesmo quando estão disponíveis os melhores métodos de acesso e estratégias de avaliação suportadas pelo SGBD. Para solucionar esse problema, por exemplo, pode-se tentar atualizar as estatísticas do SGBD e ajustar a precisão das constantes de calibração, fórmulas de cálculos de custo e parâmetros de banco; com o objetivo de tornar os cálculos dos custos das operações de execução das instruções SQL mais precisos. Entretanto, a atualização frequente das estatísticas pode degradar o desempenho do Sistema de Banco de Dados (SBD). Já a tarefa de ajustar as constantes de calibração, fórmulas de custos e parâmetros do SGBD envolve elevada complexidade e nem sempre são permitidas pelos SGBDs.

Por outro lado, dois comandos SQL com a mesma semântica, mas com sintaxes diferentes, podem apresentam tempos de resposta distintos. Este fato decorre das operações utilizadas em cada comando SQL (ordenações, agregações, remoção da valores duplicados, utilização de tabelas temporárias, subconsultas, dentre outras). Assim, variando-se as operações usadas em um comando SQL altera-se a quantidade de recursos necessários para executá-lo.

No contexto deste trabalho, considera-se que dois comandos SQL Q_1 e Q_2 são equivalentes se e somente se retornam os mesmos resultados.

Assim, uma vez que a sintaxe de um comando SQL influencia a escolha do plano de execução, o otimizador pode produzir planos de execução distintos para comandos SQL equivalentes. Consequentemente, comandos SQL equivalentes podem apresentar tempos de resposta diferentes.

Por estes motivos, uma das atividades realizadas pelo DBA (*Database Administrator*) com a finalidade de melhorar o desempenho de uma determinada consulta Q_1 consiste no ajuste (ou sintonia) de Q_1. Para isso, em geral, duas estratégias são frequentemente utilizadas: (a) reescrever o comando SQL; e (b) aplicar *Query Hinting* (SHASHA; BONNET, 2003).

Dado um determinado comando SQL Q_1, a técnica de reescrita consiste em escrever um novo comando SQL Q_2, equivalente ao comando SQL original Q_1, com o objetivo de que o tempo de resposta apresentado por Q_2, denominado TR_{Q_2}, seja menor que o tempo de resposta de Q_1, denominado TR_{Q_1}. Assim, teríamos $TR_{Q_2} < TR_{Q_1}$, o que pode ser obtido caso o plano de execução gerado pelo otimizador de consultas do SGBD para a instrução Q_2, denominado P_{Q_2}, seja mais eficiente que o plano de execução gerado para Q_1, denominado P_{Q_1}.

Com a finalidade de ilustrar os benefícios proporcionados pela reescrita de comandos SQL, considere a instrução Q_1 apresentada na Figura 1.1. Observe que é possível obter um novo comando SQL Q_2 equivalente a Q_1, removendo o operador *ALL* e incluindo a função *MIN* na subconsulta existente em Q_1. A Figura 1.2 exibe o texto do comando SQL Q_2. Os comandos Q_1 e Q_2 foram elaborados utilizando-se a base de dados do *benchmark* TPC-H e foram executados no SGBD PostgreSQL. Apesar de Q_1 e Q_2 serem sintaticamente diferentes (e apresentarem planos de execução diferentes), eles retornam o mesmo resultado. Vale destacar que a execução de Q_2 apresentou um tempo de resposta muito menor que Q_1, uma vez que $TR_{Q_1} = 13.685$ ms e $TR_{Q_2} = 1.825$ ms. Essa redução deve-se à menor quantidade de tuplas que serão materializadas durante a execução da subconsulta, visto que a subconsulta de Q_2 retorna apenas o valor mínimo da subconsulta de Q_1. As figuras 1.3 e 1.4 ilustram os planos de execução gerados pelo PostgreSQL para as instruções Q_1 e Q_2, respectivamente. Na Seção 4.1 são apresentados

```
SELECT *
FROM orders
WHERE o_totalprice < ALL (SELECT o_totalprice
                          FROM orders
                          WHERE o_orderpriority = '2-HIGH')

Tempo de execução = 13.685 ms
```

Figura 1.1: Comando SQL Q_1 (consulta que utiliza a operação sobre conjuntos ALL).

```
SELECT *
FROM orders
WHERE o_totalprice < (SELECT MIN(o_totalprice)
                      FROM orders
                      WHERE o_orderpriority = '2-HIGH')

Tempo de execução = 1.825ms
```

Figura 1.2: Consulta Q_2 (consulta reescrita a partir de Q_1).

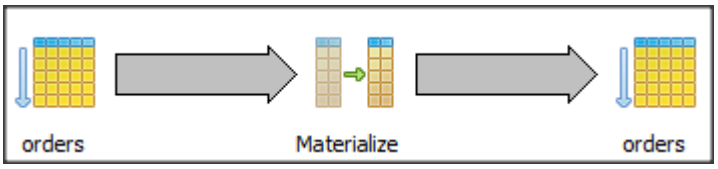

Figura 1.3: Plano de Execução P_{Q_1} gerado pelo PostgreSQL para o comando SQL Q_1

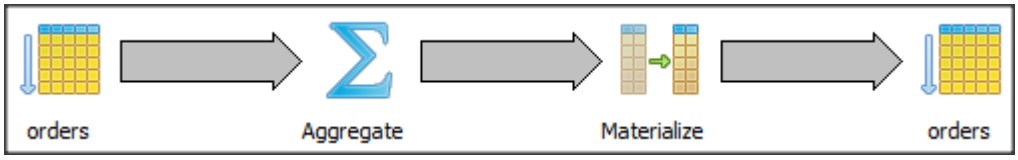

Figura 1.4: Plano de Execução P_{Q_2} gerado pelo PostgreSQL para o comando SQL Q_2.

diversos exemplos de reescrita de comandos SQL.

A aplicação de *hints* consiste em um mecanismo comumente encontrado nos sistemas de bancos de dados comerciais. Essencialmente, um *hint* instrui o otimizador a restringir seu espaço de busca para certo subconjunto de planos de execução (por exemplo, impor a escolha de planos que usem um determinado tipo índice ou determinando a ordem e/ou método de junção). O uso de um *hint* é pode ser realizado apenas concatenado-o ao comando SQL. Contudo, deve ser observado se o plano escolhido pelo otimizador para este novo comando (gerado após a inserção do *hint*) proporciona um desempenho melhor que a instrução SQL original (BRUNO; CHAUDHURI; RAMAMURTHY, 2009).

Para exemplificar os benefícios que podem ser proporcionados pela aplicação de *hints* con-

```
Select nation, o_year, sum(amount) as sum_profit
From ( Select n_name as nation, extract(year from o_orderdate) as o_year,
            l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
       From part, supplier, lineitem, partsupp, orders, nation
       Where s_suppkey = l_suppkey And ps_suppkey = l_suppkey And ps_partkey = l_partkey
            And p_partkey = l_partkey And o_orderkey = l_orderkey And
            s_nationkey = n_nationkey And p_name like '%blush%')
Group By nation, o_year
Order By nation, o_year desc
```

Figura 1.5: Comando SQL Q_3.

```
Select /*+ NO_CPU_COSTING */nation, o_year, sum(amount) as sum_profit
From ( Select n_name as nation, extract(year from o_orderdate) as o_year,
            l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
       From part, supplier, lineitem, partsupp, orders, nation
       Where s_suppkey = l_suppkey And ps_suppkey = l_suppkey And ps_partkey = l_partkey
            And p_partkey = l_partkey And o_orderkey = l_orderkey And
            s_nationkey = n_nationkey And p_name like '%blush%')
Group By nation, o_year
Order By nation, o_year desc
```

Figura 1.6: Comando SQL Q_4 (consulta gerada a partir de Q_3 com a inclusão do *hint* NO_CPU_COSTING).

sidere os comandos SQL Q_3 e Q_4 apresentados nas Figuras 1.5 e 1.6, respectivamente. Observe que o comando SQL Q_4 foi obtido a partir da inclusão do *hint* NO_CPU_COSTING no texto do comando Q_3. O comando SQL Q_3 foi elaborado utilizando-se a base de dados do *benchmark* TPC-H. Os dois comandos, Q_3 e Q_4, foram executados no SGBD Oracle. O *hint* NO_CPU_COSTING faz com que o otimizador de consultas do SGBD ignore os custos relativos ao uso da CPU. A utilização desse *hint* em Q_4 proporcionou ganhos de desempenho, uma vez que $TR_{Q_3} = 9,27$ s e $TR_{Q_4} = 4,49$ s.

Embora a sintonia de comandos SQL possa proporcionar ganhos de desempenho expressivos para as aplicações que utilizam bancos de dados relacionais, sua aplicação prática apresenta diversos desafios. Uma das tarefas diárias de um desenvolvedor de aplicações consiste em elaborar (escrever) instruções SQL que serão utilizadas nas aplicações. Em geral, esses comandos SQL são elaborados tendo como objetivo principal a recuperação dos dados desejados, sem maiores preocupações com desempenho, ou sobre como a instrução SQL será executada pelo SGBD. Além disso, os desenvolvedores frequentemente vivem pressionados por prazos e não possuem tempo suficiente para examinar em detalhes como suas instruções SQL são executadas ou mesmo testar essas instruções em um banco de dados significativo o suficiente para expor possíveis problemas de desempenho. Ademais, mesmo que um desenvolver identifique um problema de desempenho em uma determinada instrução SQL, muitas vezes ele não detém o conhecimento para solucioná-lo.

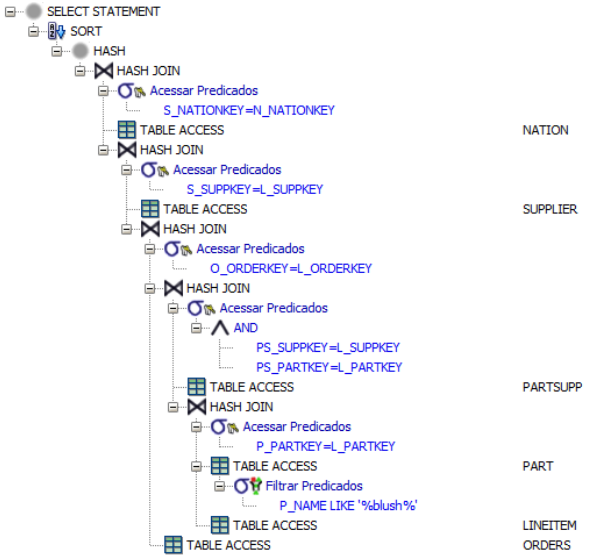

Figura 1.7: Plano de Execução P_{Q_3} gerado pelo Oracle para o comando SQL Q_3.

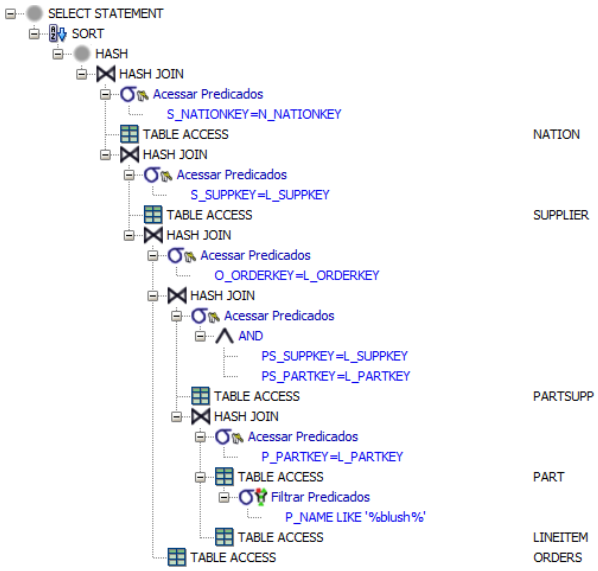

Figura 1.8: Plano de Execução P_{Q_4} gerado pelo Oracle para o comando SQL Q_4.

Neste contexto, o Administrador de Bancos de Dados (DBA) é o responsável por solucionar os problemas de desempenho apresentados na execução de comandos SQL e utiliza-se da reescrita de instruções SQL e da inserção de *hints* para ajustar (sintonizar) esses comandos. Entretanto, esse processo é complexo e requer bastante conhecimento em diversas áreas (HERODOTOU; BABU, 2009), tais como:

i otimização de consultas, execução de operadores de planos de consultas, configuração de parâmetros e outros aspectos internos dos bancos de dados;

ii identificação de índices necessários e outras estruturas de acesso;

iii manutenção de estatísticas sobre os dados; e

iv características dos sistemas de armazenamento de dados.

Além disso, em geral, o DBA trabalha de uma forma reativa, ou seja, só é acionado quando um usuário do sistema percebe que um determinado comando SQL está apresentando um desempenho insatisfatório. Depois de monitorar e identificar o problema, o DBA pode ajustar a instrução SQL e testar o novo comando em um banco de dados de teste (ou mesmo no banco em produção). Em seguida, o DBA envia o novo comando SQL ao programador a fim de que este altere o código da aplicação que acessa o banco de dados. Todo esse processo consome bastante tempo, tanto do DBA quanto do programador.

Como pode ser observado, a sintonia de instruções SQL é uma tarefa bastante complexa e que consome bastante tempo do DBA. Contudo, esta não é a única atividade desempenhada por este profissional. As atividades do DBA envolvem:

- o planejamento da capacidade de *hardware*;
- a garantia da segurança lógica dos dados;
- o projeto físico do banco de dados e
- o gerenciamento das dependências inter-sistemas (como por exemplo, entre o *middleware* e o SGBD) (MONTEIRO; LIFSCHITZ; BRAYNER, 2006).

Desta forma, fornecer ferramentas que auxiliem o DBA na difícil e repetitiva tarefa de ajustar comandos SQL torna-se de fundamental importância. Recentemente, algumas ferramentas têm sido disponibilizadas com o objetivo de auxiliar o DBA por meio de recomendações de sintonia de comandos SQL, tais como: Optim Development Studio da IBM (STUDIO, 2010), DB Optimizer XE da Embarcadero (OPTIMIZER, 2010), SQL Optimizer for Oracle da Quest (QUEST, 2010) e a Sql Tuning Advisor da Oracle (DAGEVILLE; DIAS, 2006). Contudo, em geral, essas ferramentas adotam uma abordagem *offline* (executam suas tarefas apenas quando são explicitamente acionadas pelo DBA), são específicas para um determinado SGBD e transferem para o DBA, dentre outras tarefas, a decisão de executar ou não as recomendações sugeridas.

Este trabalho propõe duas abordagens distintas para suportar a reescrita de comandos SQL em bancos de dados relacionais: uma abordagem assistida e outra automática. As duas abordagens propostas utilizam um conjunto de heurísticas para realizar a reescrita dos comandos SQL. As heurísticas são constituídas de regras que visam identificar oportunidades de sintonia nos comandos SQL. Com o objetivo de avaliar a eficiência das abordagens propostas uma avaliação experimental foi realizada. Os experimentos foram conduzidos em três diferentes cenários: i) com o *benchmark* TPC-H, ii) com a base de dados do TPC-H e uma carga de trabalho sintética e iii) com a base de dados do sistema SIG e uma carga de trabalho sintética. Para cada cenário, três SGBDs foram avaliados: PostgreSQL, Oracle e SQL Server. Os resultados dos testes realizados mostram que tanto a abordagem assistida quanto a automática proporcionaram ganhos de desempenho, reduzindo o tempo de resposta das cargas de trabalho avaliadas.

1.2 Objetivos

Com a finalidade de auxiliar o DBA na complexa tarefa de reescrever comandos SQLs, esse trabalho propõe duas abordagens distintas para a reescrita de instruções SQL em bancos de dados relacionais: uma abordagem assistida e outra automática.

As duas abordagens propostas utilizam um conjunto de heurísticas para realizar a reescrita das instruções SQL. As heurísticas são constituídas de regras para: i) identificar potenciais oportunidades de sintonia nos comandos SQL e ii) reescrever os comandos SQL com o objetivo de explorar as oportunidades identificadas. Assim, cada heurística procura identificar uma determinada oportunidade de sintonia em uma instrução SQL e, caso encontre, busca reescrever o comando SQL para explorar a oportunidade identificada.

A abordagem assistida consiste em um *advisor* capaz de:

i capturar as instruções SQL anteriormente executadas;

ii analisar essas instruções e

iii sugerir (por meio de alertas, *wizards* ou relatórios) oportunidades de sintonia (reescrita).

Desta forma, o advisor identifica instruções SQL que se fossem reescritas poderiam fazer com que o otimizador de consultas escolhesse planos de execução melhores, reduzindo o tempo de resposta dos comandos SQL. Adicionalmente, o *advisor* permite ao DBA interagir com o processo de sintonia, por exemplo, selecionando um subconjunto das heurística disponibilizadas a fim de que somente essas sejam utilizadas para reescrever as instruções SQL em geral ou um determinado comando SQL em particular. Assim, se o DBA identificar que algumas heurísticas são desnecessárias ou inadequadas ao seu banco de dados ou para uma determinada instrução SQL, ele pode desativá-las.

A abordagem automática consiste em um *middleware* que atua entre a aplicação e o SGBD. Este *middleware* é responsável por:

i receber as instruções SQL enviadas pelas aplicações;

ii analisar e reescrever as instruções SQL recebidas (se necessário);

iii enviar as instruções (reescritas ou não) para o SGBD;

iv receber do SGBD o resultado da execução de cada instrução SQL e

v envia o resultado da execução de cada comando SQL para a aplicação cliente.

A abordagem automática pode ainda utilizar as preferências definidas pelo DBA por meio *advisor*, tanto para comandos SQL em geral quanto para uma instrução SQL em particular.

As abordagens propostas apresentam ainda as seguintes características:

- São Não-intrusivas: as abordagens assistida e automática são completamente desacoplada do código-fonte do SGBD utilizado. Isso permite que a solução concebida possa ser utilizada com qualquer SGBD.
- São Independentes de localização: as duas abordagens propostas podem ser executadas em uma máquina distinta daquela utilizada para hospedar o SGBD, não consumindo recursos do servidor onde o SGBD está hospedado.
- *On-the-fly*: são executadas sempre que necessário e de forma contínua durante o funcionamento normal do SGBD.

Adicionalmente, o funcionamento da abordagem automática é completamente independente de interações com seres humanos.

1.3 Principais Contribuições

As principais contribuições desta dissertação são:

1. A identificação de um conjunto de heurísticas que podem ser utilizadas para a reescrita de instruções SQL, de forma computacional;
2. Uma abordagem para a sintonia assistida de comandos SQL;
3. Uma abordagem para a sintonia automática de instruções SQL;
4. Uma arquitetura baseadas em agentes de *Software* para a implementação da abordagem assistida;
5. Uma arquitetura baseadas em agentes de *Software* para a implementação da abordagem automática;

6. Uma implementação da abordagem assistida e da abordagem automática.As arquiteturas e as abordagens propostas foram implementadas inteiramente em linguagem Java. O protótipo implementado fornece suporte para os SGBDs PostgreSQL 8.4, Oracle 11g e SQL Server 2008. Adicionalmente, outros SGBDs podem ser facilmente adicionados ao protótipo.

1.4 Organização da Dissertação

Esta dissertação está estrutura da seguinte forma:

- No Capítulo 2, são apresentados e discutidos os conceitos fundamentais para o entendimento deste trabalho, tais como: sintonia e auto-sintonia de bancos de dados, sintonia auto-sintonia de instruções SQL e, *benchmarks*.
- No Capítulo 3, serão discutidas as principais abordagens, encontradas na literatura, para a sintonia de instruções SQL. Uma análise comparativa detalhada destas abordagens também é apresentada.
- O Capítulo 4 apresenta a pesquisa realizada com o objetivo de identificar um conjunto de heurísticas para a rescrita de consultas, de forma computacional. Adicionalmente, um estudo sobre a utilização dessas heurísticas por parte dos SGBDs comerciais é discutido.
- O Capítulo 5, apresenta as abordagens assistida e automática para a reescrita de instruções SQL. Essas abordagens são completamente desacopladas do código do SGBD utilizado e podem ser utilizadas com qualquer SGBD. Adicionalmente, a abordagem automática executa sem intervenção humana. As arquiteturas concebidas e a ferramenta implementada (suas principais características e funcionalidades, além de detalhes de implementação) são discutidas.
- O Capítulo 6 discute os testes de desempenho realizados utilizando cada uma das abordagens propostas e os resultados obtidos.
- Finalmente, o Capítulo 7 conclui este trabalho, avaliando os resultados obtidos, as dificuldades encontradas e apontando direções para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Sintonia de Bancos de Dados

O conceito de sintonia de bancos de dados refere-se à técnica de ajustar os parâmetros, configurações, estruturas de acesso e ajuste do projeto físico de um sistema de bancos de dados às necessidades de uma determinada carga de trabalho (*workload*). Por carga de trabalho, entende-se um conjunto de tarefas (consultas ou atualizações) realizadas em SGBD. Alguns autores, como (RAMALHO, 2002), defendem que, para a realização de sintonia, o DBA deve também considerar fatores externos ao SGBD, tais como o *hardware* (como a quantidade de memória, por exemplo) e o *software*(como o sistema operacional, por exemplo), antes de fazer alterações nas configurações do SGBD (MONTEIRO; LIFSCHITZ; BRAYNER, 2006).

É importante diferenciar a sintonia de banco de dados da otimização de consultas, apesar de haver uma forte relação entre as duas. A otimização de consultas é o processo realizado por um SGBD para converter um comando declarativo, escrito, por exemplo, em SQL, em um plano de execução eficiente. O otimizador é capaz de gerar planos alternativos para execução de consultas, estimar o custo de cada plano e escolher o de menor custo estimado. Para gerar os planos alternativos, o otimizador especifica todos os operadores que serão aplicados aos dados e de que forma eles serão ordenados e compostos. Além disso, o otimizador incorpora heurísticas que limitam o espaço de busca dos planos alternativos, evitando uma explosão combinatorial de planos. Assim, a otimização de consultas é um processo automático, realizado por componentes do SGBD. Já a sintonia de banco de dados é realizada, na maioria das vezes, pelo administrador do sistema (DBA) ou por um especialista em desempenho. Podemos caracterizar o desempenho de um sistema de computação como a eficiência com que este atinge seus objetivos (LIFSCHITZ; MILANES; SALLES, 2004).

A tarefa de sintonia busca atingir um melhor desempenho dos bancos de dados, fazendo com que as operações das aplicações sejam executadas em um menor tempo (tempo de resposta). Os SGBDs atuais possuem muitos de parâmetros e configurações que podem ser ajustados para alcançar um alto nível de eficiência. Tais ajustes apresentam alta complexidade, bem como as inter-relações entre esses ajustes. Na maioria dos SGBDs atuais, a sintonia é realizada de forma manual (LIFSCHITZ; MILANES; SALLES, 2004).

À medida que os bancos de dados vão se tornando mais complexos, a tarefa de sintonia fica cada vez mais difícil de ser realizada pelo DBA e impraticável ao longo do tempo. Tudo isso em razão da infinidade de configurações e ajustes que podem ser realizados pelo DBA, além da

constante verificação do funcionamento do SGBD, procurando identificar e corrigir qualquer fator que prejudique o desempenho dos sistemas (SALLES, 2004).

Todos esses fatores fizeram com que o meio acadêmico adotasse como objeto de pesquisa a realização de sintonia de banco de dados de forma automática, ou seja, de auto-sintonia. Este conceito visa diminuir a complexidade de administração dos SGBDs, bem como as tarefas manuais que o profissional DBA precisa realizar (SALLES, 2004).

2.2 Processamento de Consultas

Uma consulta expressa em uma linguagem de consulta de alto nível, tal como SQL, deve primeiro passar por uma análise léxica, uma analise sintática e ser validada. A análise léxica identifica os itens léxicos da linguagem (tais como palavras chaves da SQL, nomes de atributos e nomes relacionados) no texto da consulta, enquanto que a análise sintática verifica a sintaxe da consulta para determinar se ela está formulada de acordo com as regras sintáticas (regras gramaticais) da linguagem de consulta. A consulta também deve ser validada por meio de verificação de que todos os atributos e nomes de relacionamentos são válidos, e se são nomes com significado semântico no esquema do banco de dados específico que está sendo consultado. Então, o SGBD deve planejar uma estratégia de execução para a recuperação do resultado da consulta. Em geral, uma consulta possui muitas estratégia de execução possíveis, e o processo de escolha de uma estratégia adequada para o processamento de consulta é chamado de otimização de consulta (ELMASRI; NAVATHE, 2005).

O termo otimização é, na verdade, uma denominação imprópria porque, em alguns casos, o plano de execução escolhido não é a melhor estratégia (ótima), ela é apenas uma estratégia razoavelmente eficiente para a execução da consulta. Encontrar a estratégia ótima geralmente é uma tarefa que consome muito tempo, exceto para consultas mais simples, e pode exigir informações de como os arquivos são implementados e, até mesmo, do conteúdo dos arquivos (essas informações podem não estar completamente disponíveis no catálogo do SGBD). As duas principais técnicas para implementação da otimização de consultas são a baseada em regras heurísticas e a baseada em estimativas de custos. Geralmente as duas técnicas são combinadas em um otimizador de consultas (ELMASRI; NAVATHE, 2005).

A otimização de consultas baseada em heurísticas modifica a representação interna de uma consulta para melhorar seu desempenho, geralmente essa representação está na forma de estrutura de dados de árvore de consulta ou de um grafo de consulta. Uma das principais técnicas utilizadas por essa heurística é aplicar as operações *SELECT* e *PROJECT* antes de aplicar o *JOIN* ou outras operações binárias. Isso se deve ao tamanho do arquivo resultante de uma operação binária (tal como o *JOIN*), que geralmente é uma função multiplicativa dos tamanhos dos arquivos de entrada. As operações *SELECT* e *PROJECT* reduzem o tamanho de um arquivo e, por isso, devem ser aplicadas antes de uma junção ou outra operação binária. Em seguida, depois de modificar a representação interna da consulta, um plano de execução é gerado para executar grupos de operações como base nos caminhos de acesso disponíveis para os arquivos envolvidos na consulta (ELMASRI; NAVATHE, 2005).

A otimização baseada em custos é gerada a partir de funções estatísticas dos SGBDs. Estas funções armazenam informações referentes ao número de tuplas, aos índices, as chaves, a cardinalidade das tabelas, a distribuição dos dados nas colunas das tabelas, o uso de CPU e I/O e o tamanho das tabelas utilizadas. Como essas informações, o otimizador pode gerar vários planos de execução alternativos para uma consulta, calcular o custo de cada um deles planos e escolher o plano de menor custo. Porém, como este tipo de otimização e pode gerar um resultado que talvez não seja o melhor. Isto acontece devido ao SGBD utilizar informações estatísticas que talvez estejam desatualizadas. Assim, é necessário que as funções que armazenam essas informações sejam atualizadas periodicamente. Contudo, não é viável realizar a atualização das estatísticas todas as vezes que acontece alguma alteração no banco, pois isto implica em alto custo, mas atualizando-as periodicamente, o SGBD manterá valores próximos aos reais (ELMASRI; NAVATHE, 2005).

2.3 Sintonia Automática de Bancos de Dados

O ramo da computação autônoma que investiga a autonomia em SGBDs é denominado gerência automática de bancos de dados, ou ainda, banco de dados auto-gerenciáveis (*self-management databases*) (SALLES, 2004). A auto-sintonia ou sintonia automática de bancos de dados (*self-tuning database*) é considerada uma sub-área da gerência automática de bancos de dados que se preocupa em tornar automática a gerência de desempenho de SGBDs. Essa atividade envolve, dentre outras tarefas, o diagnóstico automático de problemas de desempenho, a gerência automática do projeto físico, a gerência automática de memória, etc. Para auxiliar a contextualização do tema, é apresentada, na Figura 2.1, parte dos ramos da computação autônoma aplicados a SGBDs. Vale destacar que os mesmos princípios, requisitos e conceitos da computação autônoma também são válidos para a sintonia automática de bancos de dados, porém, existem atividades que são específicas desse ramo da computação autônoma.

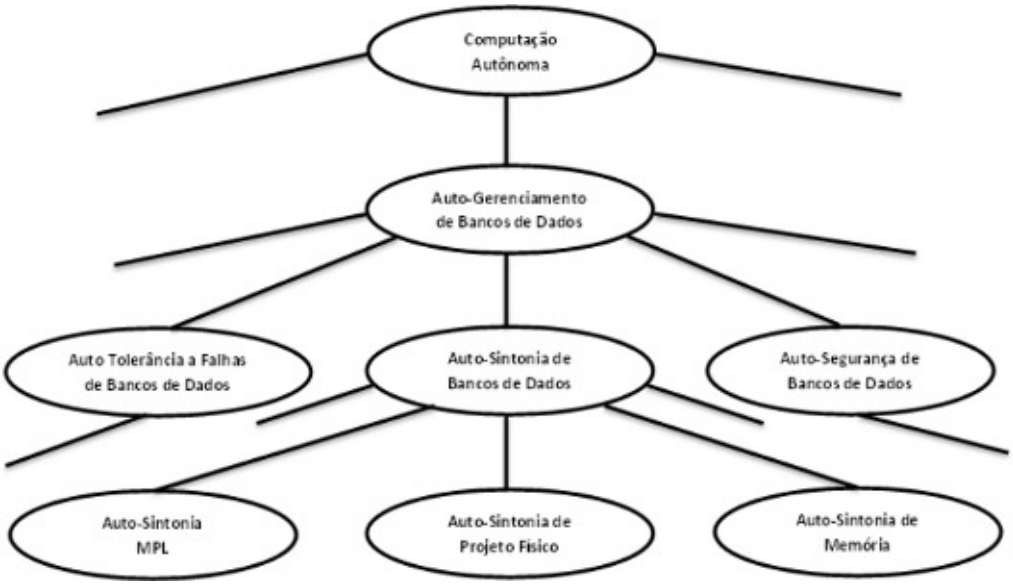

Figura 2.1: Ramos da Computação Autônoma Aplicados a SGBDs.

A relevância do estudo de auto-sintonia para a área de banco de dados aumentou significativamente nos últimos anos. Espera-se que, no futuro, os sistemas demandem uma quantidade menor de profissionais especializados para a manutenção de seu desempenho e de sua operação. Num cenário ideal, os sistemas conseguiriam alcançar um desempenho adequado sem qualquer necessidade de ajuste manual. Mais do que isto, à medida que as cargas de trabalho submetidas ao sistema mudassem, este iria procurar se adaptar dinamicamente para continuar apresentando um desempenho aceitável (LIFSCHITZ; MILANES; SALLES, 2004).

O conceito de auto-sintonia está ligado à automatização das operações do SGBD visando o aumento de seu desempenho e à redução da interação do DBA com o sistema. Essa automatização refere-se à execução e ao ajuste dos parâmetross, configurações e estruturas de um SGBD, de forma automática e com o objetivo de processar, de modo mais eficiente, uma determinada carga de trabalho submetida ao banco (LIFSCHITZ; MILANES; SALLES, 2004).

A auto-sintonia, bem como a atividade de sintonia de bancos de dados, deve respeitar alguns limites quanto ao tipo de ações que podem ser empreendidas para aumentar o desempenho do sistema. Assim, expandir a configuração de *hardware*, por meio da adição de processadores, memória e discos, ou mudar a versão do *software* do SGBD não constituem ações que estariam no escopo de um componente de auto-sintonia do sistema. Este tipo de componente deve criar controles sobre a forma como o sistema se utiliza do *hardware* disponível para acessar os dados (LIFSCHITZ; MILANES; SALLES, 2004).

Ações válidas incluem, portanto, a criação de estruturas de apoio para acelerar a busca de informações, a coleta de estatísticas sobre os dados armazenados, a alocação dos dados em disco e em memória, de forma a acelerar o seu acesso, e o controle de quantas transações serão atendidas pelo sistema e com qual tempo de resposta. Nos sistemas atuais, muitas destas decisões exigem a intervenção de DBAs especializados. O sistema não é capaz de analisar automaticamente qual é a melhor forma de utilizar o *hardware* e o *software* em quse foi instalado para processar a carga de trabalho que lhe é submetida (LIFSCHITZ; MILANES; SALLES, 2004).

Uma das grandes dificuldades da auto-sintonia de bancos de dados reside no fato de que a configuração de melhor desempenho para o sistema depende fundamentalmente do ambiente em que está inserido. A definição do ambiente varia de acordo com o foco de estudo. Para o sistema como um todo, o ambiente refere-se à combinação de *hardware*, carga de trabalho e outros programas executados no mesmo *hardware* (por exemplo, o sistema operacional). Já para um módulo específico do sistema, o ambiente pode ser definido pelos demais componentes do SGBD em contato com esse módulo e pelas requisições a que este módulo precisa atender (carga de trabalho) (LIFSCHITZ; MILANES; SALLES, 2004).

2.3.1 Classificação das Pesquisas em Auto-sintonia de Bancos de Dados

s Vários trabalhos, presentes na literatura, buscam soluções para a realização de auto-sintonia em bancos de dados. Basicamente, é possível separar os trabalhos existentes, de pesquisa em auto-sintonia, em dois grandes grupos, de acordo com o foco dado na abordagem do problema.

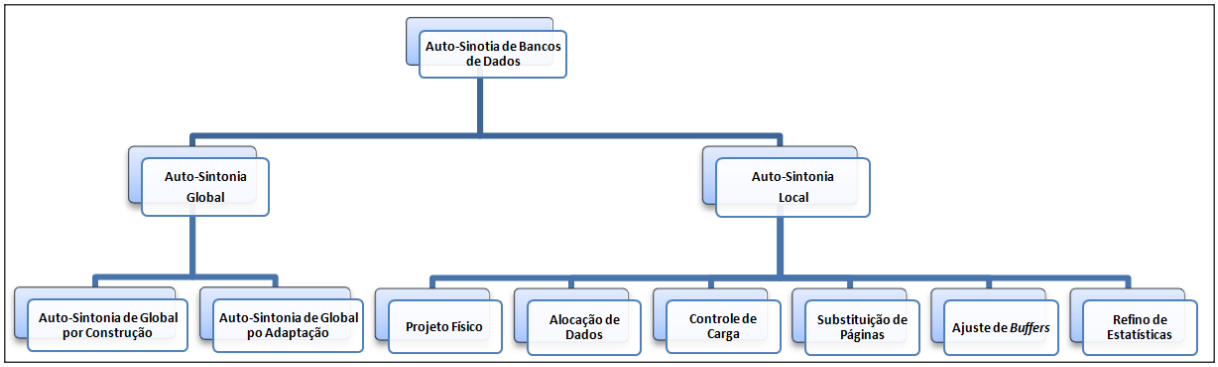

Figura 2.2: Classificação dos estudos em auto-sintonia.

A seguir, baseado em (SALLES, 2004), fornecemos uma breve descrição desses dois grupos de propostas.

No primeiro grupo estão os trabalhos de auto-sintonia local, que procuram estudar problemas específicos de sintonia existentes nos sistemas atuais. Destes problemas podemos citar o projeto físico de bancos de dados (em especial, a seleção de índices para o sistema), a alocação de dados entre diferentes elementos de armazenamento, o controle de carga, o refino das estatísticas utilizadas pelo otimizador, a substituição de páginas e o ajuste de áreas de memória. Cada um dos trabalhos de auto-sintonia local enfoca um destes problemas, analisando vantagens e desvantagens de uma solução que reduz ou elimina a intervenção humana (LIFSCHITZ; MILANES; SALLES, 2004).

No segundo grupo estão os trabalhos de auto-sintonia global, que procuram estudar como é possível tomar ações de sintonia que tragam benefícios de desempenho para o sistema como um todo. Este tipo de abordagem procura encontrar um sequilíbrio entre as diversas considerações locais de sintonia de forma a alcançar um desempenho global que seja melhor do que o que seria alcançado caso cada componente do sistema tomasse decisões de sintonia isoladamente. Um exemplo seria o de considerar ações de criação de índices em tabelas pequenas por motivos de concorrência. Ainda há poucos trabalhos nesta linha de pesquisa e, de fato, bem poucos trazem algum resultado experimental, mesmo quando consideramos trabalhos que lidam com sistemas que não são SGBDs. Isto pode estar relacionado ao fato das inter-relações entre os diversos componentes de um SGBD não serem bem conhecidas (WEIKUM et al., 2002).

Em (LIFSCHITZ; MILANéS; SALLES, 2004) é proposta uma classificação dos trabalhos na área auto-sintonia de bancos de dados relacionais. A Figura 2.2 mostra a proposta de classificação desses trabalhos.

As propostas de auto-sintonia global procuram estabelecer princípios gerais para a implementação de sistemas que se adaptem automaticamente ao seu ambiente. Além disso, esse tipo de técnica também busca manter um equilíbrio entre os vários componentes do sistema de forma a alcançar um desempenho que seja globalmente melhor do que o obtido caso cada componente tomasse as decisões de sintonia isoladamente. Os trabalhos de auto-sintonia global tendem a seguir duas possibilidades para a sua criação: por construção e por adaptação.

Os sistemas com auto-sintonia por construção são implementados de tal forma que o próprio

sistema possui o conhecimento de modelos sobre os impactos de desempenho e de sua interação com o ambiente. Assim, ele se adapta no decorrer de seu funcionamento para manter um desempenho aceitável mesmo diante de mudanças no ambiente. Uma vantagem desse tipo de abordagem é a eliminação dos parâmetros de sintonia do sistema, tornado a sua administração mais simples. Por outro lado, estes sistemas precisam lidar com a complexidade de integrar mecanismos de auto-sintonia com a base de código do próprio sistema.

Na estratégia de auto-sintonia por adaptação o estudo dos sistemas de bancos de dados é feito como uma caixa-preta e é criado um componente externo que ajusta os parâmetros e configurações de desempenho de acordo com um modelo previamente existente das interações entre o sistema e o seu ambiente. Uma vantagem desse tipo de abordagem é a possibilidade de acoplarmos componentes de auto-sintonia global a sistemas já previamente existentes sem necessidade de interferir em seu código. Uma dificuldade é a necessidade do componente criar algum tipo de modelo sobre o comportamento do sistema analisado.

A auto-sintonia local procura abordar problemas específicos de sintonia que os sistemas de bancos de dados atuais apresentam. Assim, cada trabalho busca solucionar um dos seguintes problemas:

- Projeto físico: este problema consiste em escolher a melhor organização física (índices, visões materializadas, particionamento de tabelas, etc) para uma carga de trabalho específica.
- Alocação de dados: dados N elementos de armazenamento, devemos determinar como podemos alocar e realocar dinamicamente fragmentos de arquivos ou relações nestes elementos de tal forma que o equilíbrio de carga seja o melhor possível mesmo diante de mudanças nos padrões de acesso da carga de trabalho.
- Controle de carga: em um sistema que processa múltiplas transações de forma concorrente e que se utiliza de bloqueios (*locks*) para garantia das propriedades ACID, o objetivo é regular o nível de multiprogramação do sistema de forma a evitar que a sua vazão (*throughput*) caia devido a conflitos de bloqueio (*thrashing* de contenção de dados).
- Política de substituição de páginas: o problema é manter em memória o conjunto de páginas mais populares do banco de dados, mesmo quando a carga de trabalho a que o sistema é submetido mude.
- Ajuste no tamanho de *buffers*: procura-se estudar quantos *buffers* distintos devem ser configurados no banco de dados e de que forma devem ser distribuídos os objetos (tabelas e índices) para estes buffers para que o desempenho aumente.
- Refino de estatísticas: o objetivo é escolher quais estatísticas devem ser criadas no banco de dados e refinar estas estatísticas sem necessidade de executar comandos específicos para coleta no sistema de bancos de dados.

A auto-sintonia de instruções SQL, que é um dos temas de estudo dessa dissertação, é um tipo de auto-sintonia local. Dessa forma, adicionamos a auto-sintonia de instruções SQL na

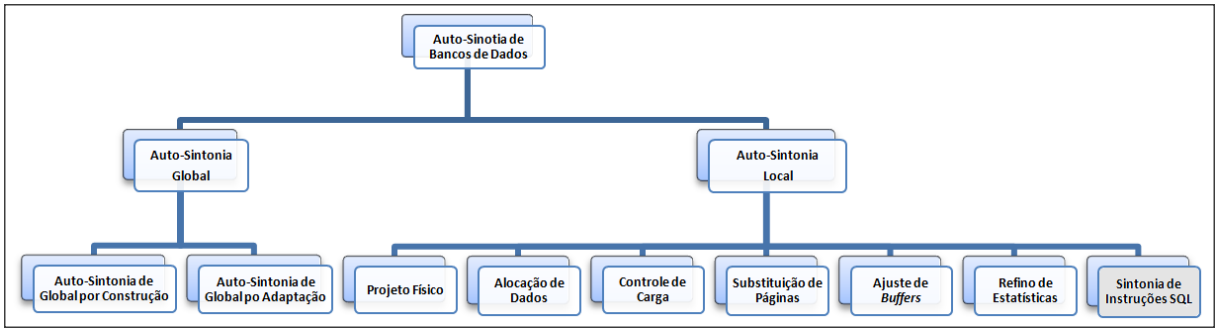

Figura 2.3: Classificação dos estudos em auto-sintonia incluindo a sintonia de comandos SQL.

classificação apresentada em (LIFSCHITZ; MILANéS; SALLES, 2004) (Figura 2.2), o que pode ser visualizado na Figura 2.3.

2.4 Sintonia de Instruções SQL

A sintonia de instruções SQL consiste na tarefa de ajustar a sintaxe dos comandos SQL com a finalidade de reduzir os tempos de resposta necessários para a execução desses comandos. Assim, dada uma determinada instrução SQL Q_1, a sintonia de instruções SQL procura ajustar a sintaxe de Q_1, gerando um novo comando SQL Q_2 equivalente a Q_1, com o intuito de melhorar o seu desempenho. Para isso, em geral, duas estratégias são frequentemente utilizadas: (a) reescrever o comando SQL; e (b) aplicar *Query Hinting* (SHASHA; BONNET, 2003).

O sucesso da sintonia de instruções SQL, geralmente, depende das habilidades do DBA. Assim, para ajustar uma instrução SQL, o DBA usa sua experiência, intuição e conhecimento. Além disso, diversas atividades estão envolvidas no processo de sintonia de comandos SQL (HERODOTOU; BABU, 2009). A seguir, descrevemos algumas dessas atividades:

1. Inicialmente, o DBA precisa criar uma réplica do ambiente de produção em um banco de dados de teste com o objetivo de assegurar que os ajustes avaliados durante os testes apresentem desempenho semelhante quando aplicados no banco de dados em produção. Por exemplo, considere um comando SQL Q_1. Assuma que o DBA realizou a sintonia de Q_1 gerando um novo comando Q_2. O DBA precisa ter a garantia de que o tempo de resposta que será necessário para a execução de Q_2 quando este for executado no banco de dados de produção será semelhante ao tempo de resposta observado durante os testes realizados no banco de dados de teste. Neste sentido, deseja-se que o plano de execução gerado pelo otimizador do SGBD quando Q_2 for executado no banco de dados de produção seja semelhante ao plano de execução gerado para Q_2 quando este foi executado no banco de dados de teste.

2. Coletar os dados monitorados e estatísticas com a finalidade de orientar o processo de ajuste do comando SQL. Essa etapa inclui obter detalhes sobre qual plano de execução foi selecionado pelo otimizador e o motivo da escolha, que índices foram utilizados, etc. O DBA, provavelmente, precisará coletar informações como valores de cardinalidade

estimados e atuais dos operadores do plano. Por exemplo, o comando *Explain* no PostgreSQL é responsável por mostrar o plano de execução escolhido juntamente com seus operadores e informações sobre cardinalidade. Com esses dados, o DBA pode analisar os motivos que fizeram o otimizador selecionar um determinado plano de execução.

3. Com base nas observações, o DBA pode formular hipóteses sobre potenciais soluções para os problemas de desempenho apresentados por um determinado comando SQL.

4. O DBA pode executar as consultas reescritas na base de dados de teste para refinar ou confirmar suas hipóteses.

5. Nem sempre o comando gerado pelo DBA, após um ajuste de sintonia, irá proporcionar um desempenho melhor que o apresentado pelo comando SQL original. Dessa forma, uma cuidadosa validação deve ser realizada para garantir que a nova instrução SQL irá realmente proporcionar ganhos de desempenho.

Fica evidente que a sintonia de instruções SQL é uma atividade bastante complexa. Além disso, é um processo que requer muito tempo do DBA e que deve ser realizado continuamente. Consequentemente, fornecer ferramentas que auxiliem o DBA na tarefa de ajustar instruções SQL torna-se de fundamental importância.

2.5 Benchmark

A avaliação de desempenho está presente em todos os momentos do ciclo de vida de um sistema computacional. Na hora de projetar, produzir ou implantar um sistema, o objetivo final é sempre o mesmo: escolher, dentre diversas opções, aquela que proporcione o melhor desempenho, com o menor custo possível. Não existe, porém, um meio universal com o qual possamos avaliar o desempenho dos diversos sistemas computacionais. Em razão da diversidade de aplicações, é necessário definir técnicas e métricas para cada caso.

Neste sentido, o termo *Benchmarking* representa um processo sistemático e contínuo de avaliação dos produtos, serviços e processos de trabalho de organizações que são reconhecidas como representantes das melhores práticas, com a finalidade de introduzir melhorias na organização; as cargas usadas nesse processo são chamadas de *benchmark*.

Benchmarks são projetados para testar características particulares dos sistemas, como, por exemplo, o poder de processamento ou o desempenho gráfico, ou ainda a capacidade do subsistema de E/S. Os *benchmarks* costumam ser classificados em sintéticos e de aplicação. *Benchmarks* sintéticos são programas que executam um conjunto reduzido de instruções, testando componentes muito específicos de um sistema. Já os *benchmarks* de aplicação simulam um ambiente real, dando uma melhor noção do desempenho do sistema como um todo. O trabalho realizado em (VIEIRA; DURãES; MADEIRA, 2005) considera algumas características que os *benchmarks* devem possuir.

1. **Representatividade:** os *benchmarks* devem representar sistemas reais, ou seja, não devem ser aplicados em sistemas simulados.

2. **Portabilidade:** os *benchmarks* devem ser portáveis para diferentes plataformas, proporcionando, assim, comparativos de desempenhos entre diversos distribuidores de sistemas.

3. **Repetibilidade:** quando um *benchmark* é aplicado no mesmo ambiente, mais de uma vez, ele deve produzir resultados semelhantes.

4. **Escalabilidade:** essa característica prevê que os *benchmarks* realizem avaliações em ambientes com diferentes capacidades.

5. **Não Intrusividade:** na necessidade de avaliar outro ambiente deve-se realizar o mínimo ou nenhuma alteração nesse novo ambiente.

6. **Simplicidade:** os *benchmarks* devem ser de fácil implementação e utilização pelos usuários que farão as avaliações.

O surgimento do *benchmarking* se deu no começo da década de 1980. Naquela época, os processadores eram os componentes mais caros dos sistemas, de maneira que se costumava denotar o desempenho do sistema como o desempenho do próprio processador. Foram desenvolvidos, então, os primeiros programas sintéticos, com o intuito de testar a capacidade de processamento dos sistemas. Exemplos da primeira geração de *benchmarks* incluem o *Dhrystone* (desempenho aritmético com inteiros), *Whetstone* (desempenho aritmético de ponto flutuante) e *Linpack* (operações com sistemas de equações lineares).

Com o passar dos anos, no entanto, ficou claro que não era suficiente medir o desempenho somente do processador. Além do mais, era comum os fabricantes realizarem otimizações nos compiladores específicos para os conjuntos de instruções desses *benchmarks*, o que gerava resultados distantes dos obtidos em aplicações do mundo real. Essa falta de credibilidade ocasionou o surgimento das primeiras organizações de *benchmark*, entidades neutras, sem fins lucrativos, que normalmente têm como membros os grandes fabricantes de *hardware* e *software*. Dentre elas, as mais conhecidas são a SPEC (*Standard Performance Evaluation Corporation*), e o TPC (*Transaction Processing Performance Council*), criado em 1988. A SPEC é responsável por diversos *benchmarks*, voltados para diversas áreas, dentre elas desempenho de CPU, de servidor *Web*, de servidor de banco de dados, de aplicações cientificas, etc. Já o TPC mantém *benchmarks* voltados ao desempenho de sistemas de bancos de dados. Essas organizações ajudaram a recuperar a credibilidade dos *benchmarks*, visto que a publicação de resultados é sujeita a regras rígidas, passando, inclusive, por procedimentos de auditoria.

O TPC tem por objetivo estabelecer critérios de performance de processamento de transações e de bancos de dados por meio de *benchmarks*, ou testes para estabelecer padrões de referência (tais como o TPC-C, o TPC-W e o TPC-H), e divulgar os dados reais dessa performance com base nos testes realizados. Os *benchmarks* TPC são submetidos a exigências extremamente rigorosas, principalmente nos quesitos confiabilidade e durabilidade e são sempre aplicados perante uma auditoria independente. Os membros desse Conselho incluem as principais empresas

de bancos de dados e fornecedores de sistemas de *hardware* do mercado - HP, IBM, Oracle, Microsoft, Unisys, Sun, Intel, AMD, Dell, Fujitsu, NEC, Teradata, Novell, Sybase, Bull, Netezza. As empresas participam dos *benchmarks* TPC para demonstrar, objetivamente, a performance de seus sistemas em um ambiente controlado, e para aplicar as tecnologias utilizadas durante o processo de testes à criação de produtos de *hardware* e de *software* ainda mais robustos e escaláveis. Porém, o TPC não disponibiliza um "*toolkit*" (*kit* de ferramentas) que possibilite ou facilite os testes de performance.

Já a organização OSDL (*Open Source Development Labs*) foi fundada em 2000 e mantém um conjunto de testes de desempenho para avaliar o comportamento de soluções de *software* livre, em especial, o sistema operacional Linux. Na verdade, ela fornece o "estado da arte" em computação e ambientes de teste para acelerar o crescimento e adoção do S.O. Linux nas empresas. Existem testes voltados especificamente para sistemas de bancos de dados e inspirados nos *benchmarks* do TPC, que são os *toolkits* DBT-1 (*Database Test 1*), DBT-2 (*Database Test 2*), DBT-3 (*Database Test 3*) e DBT-4 (*Database Test 4*). A OSDL recebe investimentos de grandes empresas como Fujitsu, HP, IBM e Intel. Todavia, estes *toolkits* somente podem ser utilizados com os SGBDs PostgreSQL e MySQL, sob o sistema operacional Linux.

Nesta dissertação, utilizamos o *benchmark* TPC-H (Apêndice B). O TPC-H é um *benchmark* de suporte a decisões que consiste em um conjunto de consultas *ad-hoc* voltadas para os negócios e modificações de dados simultâneas. Tem por finalidade simular e avaliar o desempenho de um ambiente de *Data Warehouse* (OLAP - *On-Line Analytical Processing*).

3 TRABALHOS RELACIONADOS

Em (KRISHNAPRASAD et al., 2004) é mostrada uma técnica de otimização de consultas XML utilizando bancos de dados relacional ou objeto-relacional. A ideia é transparentemente transformar, em tempo de compilação, uma consulta XML em seu equivalente relacional ou objeto-relacional por meio de técnica de reescrita, de modo que um otimizador clássico possa otimizar ainda mais a consulta e um mecanismo de execução orientado a tupla possa executá-lo de forma eficiente, escolhendo a melhor ordem de junção e/ou índices nas tabelas, por exemplo.

No trabalho de (BRUNO; CHAUDHURI; RAMAMURTHY, 2009) é proposta um *framework* chamado de *Power Hints* que é utilizado para auxiliar o DBA na inserção de *hints*. Com *hints* podemos orientar o otimizador a escolher um determinado tipo de plano que melhore a execução da consulta, restringindo o espaço de busca dos planos de execução. Entretanto, não é fácil utilizar *hints* para expressar certas restrições de estruturas de um plano de execução. E a granularidade dessas restrições de *hints* também podem não ser tão finas o suficiente para uma sintonia mais precisa, tornado o trabalho mais complexo até para DBAs experientes. Com o *framework* proposto, *hints* podem ser especificados utilizando expressões regulares, tornando mais fácil criar restrições para criação de planos de execução de consulta.

Em (HERODOTOU; BABU, 2009) é implementada uma ferramenta chamada de zTuned para automatizar os experimentos de sintonização SQL em bancos de dados, liberando usuário ou DBA dessa atividade. Essa ferramenta utiliza os mecanismos de estimativas de custos de plano de execução de consultas nos bancos de dados, como o comando Explain do PostgreSQL responsável por mostrar um plano de execução juntamente com todos seus operadores e informações úteis sobre eles como: (a) tempo estimado antes da primeira *tupla* ser retornada, (b) tempo total estimado para todas as *tuplas* serem retornadas e (c) o número estimado de *tuplas* a serem retornadas. Entretanto, os autores acharam o comando Explain no PostgreSQL não adequando o suficiente para a ferramenta, então implementaram outro comando chamado Explain_Plan que, além de possuir o que Explain já faz, possibilita fornecer valores de cardinalidade como entrada e testar planos de execução para comparar valores estimados e reais de tempo de execução e de cardinalidade. A ferramenta produz planos que possuem operadores com a mesma cardinalidade do plano gerado pelo otimizador (chamados no trabalho de planos da vizinhança) e usando o mecanismo de estimativa de custos do banco (como o comando Explain_Plan no PostgreSQL, por exemplo) escolhe o plano ótimo da vizinhança. É utilizado também o que foi chamado de inter-transformações para produzir outras vizinhanças. Então os planos ótimos de cada vizinhança são comparados para se obter o melhor. A geração de planos ocorre independente do otimizador de consultas, permitindo uma grande e diferente exploração de planos. A ferramenta trabalha desacoplada do otimizador, e pode ser potencialmente utili-

zada com qualquer banco de dados que use um otimizador baseado em custos.

Em (DAGEVILLE; DIAS, 2006) é descrita a abordagem introduzida no Oracle 10g para sintonia automática. Um dos seus principais componentes é o Automatic SQL Tuning Advisor que fornece recomendações de ajuste de instruções SQL, abrangendo otimização de consultas, análise de caminho de acesso e reescrita de consultas. Esse *advisor* é implementado como um núcleo de aprimoramento do otimizador de consultas. Nele é utilizado o conceito chamado de SQL Profiling que denota a capacidade do otimizador obter informações auxiliares sobre uma SQL baseadas em (i) verificar se uma estatística está faltando ou desatualizada; (ii) testar e validar estimativas padrões necessárias, guardando erros se forem encontrados; e (iii) analisar e determinar a melhor configuração de parâmetros do banco de dados baseado no histórico de execução de uma instrução SQL. Esse componente pode melhorar a performance de consultas através de recomendações de criação de índices baseado em predicados e cláusulas presentes na SQL, e recomendações de oportunidades de reescrita de consulta utilizando equivalências semânticas entre consultas diferentes que retornem o mesmo resultado.

Existem algumas ferramentas no mercado, como IBM Optim Development Studio [Optim Studio], Embarcadero DB Optimizer XE [Embarcadero 2010] e Quest SQL Optimizer for Oracle [Quest 2010], que já buscam solucionar o problema de baixo desempenho de consultas através da sintonia de instuções SQL. Tais ferramentas permitem ao DBA identificar e otimizar consultas através de recomendações de reescrita. Contudo, elas adotam uma abordagem *offline* na solução do problema e transferem para o DBA, dentre outras tarefas, a decisão de executar ou não as recomendações sugeridas. Todavia, em ambientes dinâmicos, com consultas *ad-hoc*, torna-se bastante complexa a atividade de reescrita de consultas. Fica evidente que essas ferramentas possuem limitações importantes: serem dependentes de intervenções humanas e não executarem de forma contínua, além do fato de serem soluções intrusivas (acopladas ao código-fonte do SGBD). A Tabela 3.1 sintetiza as características presentes nessas ferramentas a fim de mostrar possíveis pontos que a ferramenta proposta visa solucionar.

A ferramenta IBM Optim Development Studio permite capturar métricas de desempenho no DB2, como com que freqüência as instruções SQL são executadas e por quanto tempo, tempo decorrido das instruções (tempos total, mínimo, máximo e médio) e quantas vezes cada instrução é executada. Também possibilita classificar consultas por custo a fim de localizar rapidamente as mais dispendiosas. Com a combinação custo, tempo decorrido e número de execuções; fica mais fácil decidir em que consultas concentrar os esforços de ajuste. Além disso, o Optim Development Studio apresenta recursos de visualização para que DBAs e desenvolvedores recebam recomendações sobre como reescrever uma consulta de modo a melhorar sua eficiência com base nas regras de boas práticas. As recomendações para alterar as consultas baseiam-se na premissa de que as estatísticas do catálogo do DB2 estão precisas.

O Embarcadero DB Optimizerb XE é uma ferramenta heterogênea para desempenho de bancos de dados e aplicativos capaz de descobrir, diagnosticar e otimizar SQLs com baixo desempenho. Essa ferramenta provê a inserção de *hints* como maneira de direcionar o otimizaror a escolher um determinado plano de execução mais eficiente e a reescrita de consultas com o objetivo de eliminar junções cartesianas, *outer join* inválido e transitividade. Outro recurso fornecido é a analise de índices, que permite aos DBAs e desenvolvedores terem uma melhor visão de quais

Tabela 3.1: Comparativo das características presente nas ferramentas de reescrita.

	SGBD	Automática	*On-the-fly*	Sintoniza Índices	Rescreve Consultas
Optim Development Studio	DB2	Não	Não	Não	Sim
DB Optimizer	Independente	Não	Sim	Sim	Não
SQL Optimizer for Oracle	Oracle	Não	Não	Sim	Não
Sql Tuning Advisor	Oracle	Não	Sim	Sim	Sim
Ferramenta Proposta	Independente	Sim	Sim	Não	Sim

índices são usados, quais não são e quais estão faltando. No caso de estarem faltando índices, a ferramenta oferece recomendações de criação de índices para melhorar a performance. Além disso, a ferramenta possui uma opção avançada de monitoramento contínuo que fornece alertas, notificações e expressões personalizadas.

A ferramenta Quest SQL Optimizer for Oracle ajuda a identificar problemas em códigos SQL. Ela varre o código em busca de indicadores que levem a problemas de desempenho, reescreve a SQL e testa o desempenho. Com essa ferramenta os DBAs também podem gerenciar o desempenho da SQL após alterações no banco de dados, tais como criação do índice, alterações de configuração, *upgrades*, migrações e outros. Os DBAs podem analisar e comparar planos de execução de várias instruções SQL em ambientes de banco de dados diferentes para identificar variações e degradação de desempenho.

Em (BRUNO, 2001), o autor defende a ideia de que as sessões de sintonia (mais especificamente de sintonia do projeto físico) devem ser altamente interativas, possibilitando ao DBA experimentar e validar suas escolhas de uma maneira ágil e rápida. Além disso, a participação efetiva do DBA poderia contribuir para se alcançar soluções melhores, uma vez que sua experiência e seu conhecimento poderiam influenciar e conduzir o processo de sintonia. Adicionalmente, as sessões interativas de sintonia teriam o potencial de mudar forma com os DBAs raciocinam sobre o problema em questão.

Nesta mesma linha de pensamento, o trabalho apresentado em (MAIER et al., 2010; ALAGIANNIS et al., 2010) propõe uma solução interativa, denominada PARINDA (*PARtition and INDex Advisor*), para a sintonia do projeto físico de bancos de dados relacionais. A ferramenta PARINDA foi implementada para o SGBD PostgreSQL e funciona da seguinte maneira: dada uma carga de trabalho contendo um conjunto de consultas SQL, o PARINDA permite que o DBA sugira manualmente um conjunto de índices candidatos e a ferramenta mostra visualmente os benefícios proporcionados pelos índices escolhidos, bem como a interação entre eles. Além disso, a ferramenta pode sugerir o melhor momento para se criar fisicamente os índices sugeridos. Finalmente, o PARINDA pode monitorar continuamente o desempenho do SGBD diante da carga de trabalho a ele submetida e sugeriri automaticamente novas estruturas de índices com a finalidade de melhorar o desempenho da carga de trabalho, reduzindo o seu

tempo de execução.

Bruno et al propõem em (BRUNO; CHAUDHURI, 2010) uma ferramenta interativa voltada para o *Microsoft SQL Server*. A ferramenta proposta é similar ao PARINDA uma vez que possibilita sessões de sintonia interativas, permitindo ao DBA testar diferentes opções de sintonia e observar o comportamento resultante de suas escolhas.

Uma nova técnica para a recomendação de índices, denominada sintonia semi-automática, é proposta em (SCHNAITTER; POLYZOTIS, 2012). Esta técnica baseia-se na ideia de inserir o DBA no processo de sintonia, gerando recomendações que utilizam as preferências e o *feedback* do DBA. A abordagem proposta também monitora continuamente a carga de trabalho submetida ao SGBD e deixa a decisão final sobre a criação de índices para o DBA.

Apesar dos trabalhos apresentados em (BRUNO, 2001; BRUNO; CHAUDHURI, 2010; MAIER et al., 2010; ALAGIANNIS et al., 2010; SCHNAITTER; POLYZOTIS, 2012) não estarem diretamente relacionados ao problema da sintonia de instruções SQL, estes introduziram a ideia de sintonia assistida (ou interativa ou ainda semi-automática). A ideia de sintonia assistida foi utilizada em uma das abordagens propostas nesta dissertação para a sintonia de instruções SQL.

4 HEURÍSTICAS PARA SINTONIA DE INSTRUÇÕES SQL

4.1 Descrição das Heurísticas Investigadas

Existem diversas situações comuns onde o otimizador não consegue obter planos de execução ótimos devido a estruturas presentes nas instruções SQL, como, por exemplo: DISTINCT e GROUP BY desnecessários, funções e expressões aritméticas aplicadas a colunas que possuem índices, operações sobre conjuntos (ANY, SOME, ALL e IN) que poderiam ser evitadas, etc.

Neste trabalho, após uma ampla pesquisa bibliográfica, essas situações foram identificadas, analisadas e documentadas. Em seguida, um conjunto contendo as heurísticas para a reescrita de instruções SQL, que poderiam ser aplicadas de forma computacional, foi selecionado. As heurísticas encapsulam o conhecimento dos especialistas em sintonia de instruções SQL. Cada heurística é capaz de identificar uma determinada oportunidade de sintonia em um comando SQL e reescrevê-lo, gerando uma nova instrução SQL, de forma a explorar a oportunidade observada. A Tabela 4.1 resume as heurísticas para sintonia de instruções SQL identificadas e utilizadas nessa dissertação.

As heurísticas H2, H3, H4 e H9 aplicam-se exclusivamente a consultas. As sete heurísticas restantes (H1, H5, H6, H7, H8, H10 e H11) podem ser aplicadas em instruções SQL referentes a consultas, atualizações e exclusões. Para cada uma dessas sete heurísticas foi necessário implementar três versões diferentes (uma versão para avaliar as consultas, outra voltada para os comandos de atualização e uma terceira versão para avaliar os comandos de exclusão). Desta forma, o número de heurísticas efetivamente implementadas passou de onze para vinte e seis. A seguir, discutiremos o funcionamento de cada uma das heurísticas apesentadas na Tabela 4.1. Além disso, o funcionamento de cada uma dessas heurísticas será discutido e ilustrado por meio de exemplos. As consultas utilizadas nesses exemplos foram elaboradas utilizando-se a base de dados do *benchmark* TPC-H e foram executadas no SGBD PostgreSQL. Todas as heurísticas utilizadas nesse trabalho são idempotentesD, ou seja, as heurísticas retornam os mesmos resultados para qualquer SGBD. colunas

4.1.1 Heurística H1

A heurística H1 consiste em reescrever duas instruções SQL Q_1 e Q_2 em uma única instrução SQL Q_3, onde:

- A instrução Q_1 cria uma tabela temporária.

Tabela 4.1: Heurísticas para reescrita de instruções SQL.

	Heurísticas para reescrita de instruções SQL
H1	Trocar consultas que criam e utilizam tabela temporária por uma subconsulta equivalente.
H2	Eliminar GROUP BY desnecessário.
H3	Mover cláusulas desnecessárias do HAVING para o WHERE.
H4	Trocar cláusulas com disjunção no WHERE por união de consultas.
H5	Retorna apenas o resultado máximo ou mínimo de uma subconsulta em uma operação sobre conjunto ALL.
H6	Retorna apenas o resultado máximo ou mínimo de uma subconsulta em uma operação sobre conjunto SOME.
H7	Retorna apenas o resultado máximo ou mínimo de uma subconsulta em uma operação sobre conjunto ANY.
H8	Trocar operação de conjunto IN por uma junção.
H9	Eliminar DISTINCT desnecessário.
H10	Modificar função aplicada em uma coluna com índice para outra expressão equivalente sem uma função aplicada em uma coluna com índice.
H11	Mover expressão aritmética aplicada a uma coluna com índice para outra posição na expressão.

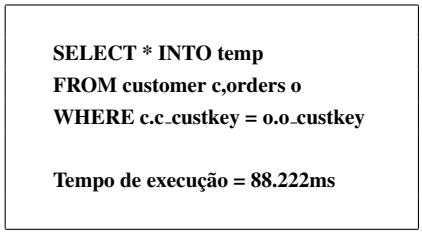

Figura 4.1: Comando SQL Q_1 (consulta que cria uma tabela temporária denominada “temp”).

- A instrução Q_2 utiliza a tabela temporária criada por Q_1.

A ideia central da heurística H1 consiste em substituir a referência à tabela temporária (utilizada por Q_2) por uma subconsulta (de código semelhante a Q_1). As Figuras 4.1, 4.2 e 4.3 ilustram um exemplo de aplicação da heurística H1. A Figura 4.1 mostra uma consulta Q_1, a qual cria uma tabela temporária denominada “temp”. A Figura 4.2 exibe uma consulta Q_2 que utiliza a tabela temporária “temp” criada por Q_1. A Figura 4.3 ilustra a consulta Q_3, a qual é gerada após a aplicação da heurística H1, tendo Q_1 e Q_2 como entrada. A consulta Q_3 apresenta ganhos de desempenho, ou seja, o tempo de resposta de Q_3 é menor que a soma dos tempos de resposta de Q_1 e Q_2. O ganho de desempenho apresentado por Q_3 decorre do fato de não ser mais necessária a criação física de uma tabela temporária. A heurística H1 foi identificada a partir de (ELMASRI; NAVATHE, 2005; SHASHA; BONNET, 2003; RAMAKRISHNAN; GEHRKE, 2008).

```
SELECT c_name, AVG(o_totalprice)
FROM temp
GROUP BY c_name

Tempo de execução = 31.140ms
```

Figura 4.2: Comando SQL Q_2 (consulta que utiliza a tabela temporária "temp").

```
SELECT c_name, AVG(o_totalprice)
FROM (SELECT *
        FROM customer AS c, orders AS o
        WHERE c.c_custkey = o.o_custkey) AS temp
GROUP BY c_name

Tempo de execução = 33.384ms
```

Figura 4.3: Comando SQL Q_3 (consulta gerada a partir de Q_1 e Q_2, após a aplicação da heurística H1).

4.1.2 Heurística H2

A existência de um operador *GROUP BY* desnecessário em uma consulta Q diminui o desempenho de Q (aumentando o seu tempo de resposta) uma vez que força o ordenamento dos dados, o que envolve um elevado custo computacional de I/O. Contudo, alguns otimizadores implementam o comando *GROUP BY* através do uso de *hash*, mas, da mesma forma, essa operação pode tornar a consulta mais lenta, uma vez que é uma operação adicional a ser realizada. Por exemplo, o Oracle e o SQL Server implementa o *GROUP BY* utilizando *hash*, enquanto que o PostgreSQL utiliza ordenação. Além disso, nem sempre os dados retornados pela consulta, após aplicação dessa heurística, podem estar na mesma ordem dos dados retornados pela consulta original, apesar dos dados retornados nas duas consultas serem os mesmos. A heurística H2 busca remover agrupamentos desnecessários, os quais podem ser identificados quando:

1. A cláusula *SELECT* de uma instrução SQL Q contém apenas uma coluna, e essa coluna possui uma função de agregação. A Figura 4.4 ilustra um exemplo desse caso.

2. O comando *SELECT* de uma instrução SQL Q não contém funções de agregação e um dos atributos no *GROUP BY* é uma chave primária, em duas situações:

   - A instrução Q possui apenas uma tabela. Supondo que *c_custkey* seja a chave primária da tabela "*customer*", a Figura 4.5 mostra um exemplo dessa situação.
   - A instrução Q possui mais de uma tabela e a chave primária de uma das tabelas não é chave estrangeira em nenhuma das demais tabelas de Q. Essa situação será detalhada a seguir.

```
SELECT MAX(c_custkey)
FROM customer
GROUP BY c_custkey
```

Figura 4.4: Exemplo 1 (primeiro caso onde é possível remover o operador *GROUP BY*).

```
SELECT c_custkey, c_name
FROM customer
GROUP BY c_custkey, c_name
```

Figura 4.5: Exemplo 2 (segundo caso onde é possível remover o operador *GROUP BY*).

```
SELECT c_custkey, c_name, c_address, c_phone
FROM customer, nation, region
WHERE c_nationkey = n_nationkey AND n_regionkey = r_regionkey
GROUP BY c_custkey, c_name, c_address, c_phone

Tempo de execução = 8.397ms
```

Figura 4.6: Comando SQL Q_4 (consulta envolvendo um *GROUP BY* desnecessário).

```
SELECT c_custkey, c_name, c_address, c_phone
FROM customer, nation, region
WHERE c_nationkey = n_nationkey AND n_regionkey = r_regionkey

Tempo de execução = 7.040ms
```

Figura 4.7: Comando SQL Q_5 (consulta gerada a partir de Q_4, após a aplicação da heurística H2).

As Figuras 4.6 e 4.7 mostram um exemplo de aplicação da heurística H2. A Figura 4.6 mostra uma consulta Q_4, a qual possui um operador *GROUPBY* desnecessário. A Figura 4.7 exibe uma consulta Q_5, que é gerada após a aplicação da heurística H2, tendo Q_4 como entrada. A consulta Q_5 apresenta ganhos de desempenho, ou seja, o tempo de resposta de Q_5 é menor que o tempo de resposta de Q_4. O ganho de desempenho apresentado por Q_5 decorre do fato de não ser mais necessária uma operação de ordenação. A heurística H2 foi identificada a partir de (ELMASRI; NAVATHE, 2005; SHASHA; BONNET, 2003).

Vale destacar que a heurística H2 somente remove um operador *GROUP BY* de uma instrução SQL Q se Q não possui um operador *HAVING*. Caso contrário, não é possível remover o operador *GROUP BY*. A Figura 4.8 apresenta uma instrução SQL Q_6 que ilustra esse caso.

```
SELECT c_custkey, c_name, c_address, c_phone
FROM customer, nation, region
WHERE c_nationkey = n_nationkey AND n_regionkey = r_regionkey
GROUP BY c_custkey, c_name, c_address, c_phone
HAVING avg(n_nationkey) > max(n_nationkey)/2

Tempo de execução = 7.500ms
```

Figura 4.8: Comando SQL Q_6 (consulta onde o operador *GROUP BY* não pode ser removido devido a existência do operador *HAVING*).

```
SELECT o_orderkey, o_custkey, o_orderpriority, o_clerk
FROM orders
GROUP BY o_orderkey, o_custkey, o_orderpriority, o_clerk
HAVING o_orderpriority = '5-LOW'

Tempo de execução = 11.430ms
```

Figura 4.9: Comando SQL Q_7 (consulta envolvendo uma cláusula *HAVING* desnecessária).

4.1.3 Heurística H3

A heurística H3 procura remover uma cláusula *HAVING* desnecessária. Essa fato ocorre quando uma instrução SQL *Q* possui uma cláusula *HAVING* mas esta não envolve funções de agregação (como por exemplo, *MAX* ou *AVG*). Nestes casos, o predicado utilizado na cláusula *HAVING* pode ser movido para a cláusula *WHERE*.

As Figuras 4.9 e 4.10 ilustram um exemplo de aplicação da heurística H3. A Figura 4.9 mostra uma instrução SQL Q_7, que possui uma cláusula *HAVING* desnecessária, ou seja, que não envolve funções de agregação. A Figura 4.10 exibe uma instrução SQL Q_8, que é gerada após a aplicação da heurística H3, tendo Q_7 como entrada. A consulta Q_8 apresenta ganhos de desempenho, ou seja, o tempo de resposta de Q_8 é menor que o tempo de resposta de Q_7. A heurística H3 foi identificada a partir de (SHASHA; BONNET, 2003; RAMAKRISHNAN; GEHRKE, 2008).

Além disso, vale destacar que sobre Q_8 pode-se ainda aplicar a heurística H2 (na tentativa de remover um *GROUP BY* desnecessário), o que geraria a instrução SQL Q_9, a qual é exibida na Figura 4.11. A heurística H3 foi identificada em (SHASHA; BONNET, 2003; RAMAKRISHNAN; GEHRKE, 2008).

4.1.4 Heurística H4

A utilização de disjunções na cláusula WHERE pode induzir o otimizador a gerar planos de execução que não utilizem as estruturas de índices disponíveis. Nestes casos, a consulta pode

```
SELECT o_orderkey, o_custkey, o_orderpriority, o_clerk
FROM orders
WHERE o_orderpriority='5-LOW'
GROUP BY o_orderkey, o_custkey, o_orderpriority, o_clerk

Tempo de execução = 10.802ms
```

Figura 4.10: Comando SQL Q_8 (consulta gerada a partir de Q_7, após a aplicação da heurística H3).

```
SELECT o_orderkey, o_custkey, o_orderpriority, o_clerk
FROM orders
WHERE o_orderpriority = '5-LOW'

Tempo de execução = 9.072ms
```

Figura 4.11: Comando SQL Q_9 (consulta gerada a partir de Q_8, após a aplicação da heurística H2).

```
SELECT l_orderkey
FROM lineitem
WHERE l_quantity < 20 or l_linenumber > 5

Tempo de execução = 10.729ms
```

Figura 4.12: Comando Q_{10} (consulta contendo uma disjunção na cláusula *WHERE*).

ser reescrita utilizando o operador *UNION*. Para ilustrar a heurística H4, considere a existência de duas estruturas de índices i_1 e i_2 definidas, respectivamente, sobre so atributos *l_quantity* e *l_linenumber* da tabela *lineitem*. A Figura 4.12) mostra uma consulta Q_{10} que contém uma disjunção na cláusula *WHERE* (*l_quantity* $<$ 20 or *l_linenumber* $>$ 5). Neste caso, em alguns SGBDs, o otimizador de consultas gera um plano de execução que não utiliza os índices i_1 e i_2. Para solucionar esse problema a heurística H4 reescreve a consulta Q_{10}, gerando uma nova consulta Q_{11} a partir da união de duas subconsultas. Cada uma das duas subconsultas deve possuir os mesmos atributos na cláusula *SELECT*, as mesmas tabelas na cláusula *FROM*, mas apenas um dos predicados de seleção da cláusula *WHERE*. A Figura 4.13 mostra a sintaxe da consulta Q_{11}. A heurística H4 foi identificada em (ELMASRI; NAVATHE, 2005; SHASHA; BONNET, 2003; RAMAKRISHNAN; GEHRKE, 2008).

```
SELECT l_orderkey
FROM lineitem
WHERE l_quantity < 20
UNION
SELECT l_orderkey
FROM lineitem
WHERE l_linenumber > 5

Tempo de execução = 9.671ms
```

Figura 4.13: Comando Q_{11} (consulta gerada a partir de Q_{10}, após a aplicação da heurística H4).

```
SELECT *
FROM part
WHERE p_retailprice > ALL(SELECT p_retailprice
                          FROM part
                          WHERE p_container = 'JUMBO CAN')

Tempo de execução = 6.816ms
```

Figura 4.14: Comando SQL Q_{12} (consulta contendo um operador sobre conjuntos *ALL* desnecessário).

4.1.5 Heurística H5

A heurística H5 busca remover o operador sobre conjuntos *ALL* quando este é desnecessário. O operador *ALL* permite a uma consulta externa fazer comparações usando relacionais ($=, <, <=, >, >=, <>$) com os elementos de um conjunto retornado por uma subconsulta. Este operador devolve *TRUE* se todas as linhas do conjunto satisfazem a condição, ou seja, devolve *FALSE* se alguma linha pertencente ao conjunto não a satisfaz. Um exemplo da aplicação da heurística H5 é apresentado nas Figuras 4.14 e 4.15. O comando SQL Q_{12} (Figura 4.14) seleciona todas as linhas da tabela "*part*" que possuem o preço a varejo ("*p_retailprice*") maior que todos os elementos retornados pela subconsulta. Observe que o operador *ALL* pode ser removido caso a função *MAX* seja inserida na subconsulta. Assim, ao invés de comparar "*p_retailprice*" como todos os elementos retornados pela subconsulta, "*p_retailprice*" seria comparado com apenas um elemento, uma vez que agora a subconsulta retorna apenas um elemento (o valor máximo de *p_retailprice*). O comando SQL Q_{13} (Figura 4.15) é gerado pela aplicação da heurística H5 no comando SQL Q_{12} (Figura 4.14). A heurística H5 foi identificada a partir de (ELMASRI; NAVATHE, 2005).

4.1.6 Heurística H6

A heurística H6 busca remover o operador sobre conjuntos *SOME* quando este é desnecessário. O operador *SOME* permite a uma consulta externa fazer comparações usando relacionais ($=, <$

```
SELECT *
FROM part
WHERE p_retailprice > (SELECT MAX(p_retailprice)
                       FROM part
                       WHERE p_container='JUMBO CAN')

Tempo de execução = 140ms
```

Figura 4.15: Comando SQL Q_{13} (consulta gerada após a aplicação da heurística H5 sobre Q_{12}).

```
SELECT o_orderkey, o_custkey, o_orderstatus, o_totalprice
FROM orders
WHERE o_totalprice > SOME (SELECT o_totalprice
                           FROM orders
                           WHERE o_orderpriority = '2-HIGH')

Tempo de execução = 49.880ms
```

Figura 4.16: Comando SQL Q_{14} (consulta contendo um operador sobre conjuntos *SOME* desnecessário).

```
SELECT o_orderkey, o_custkey, o_orderstatus, o_totalprice
FROM orders
WHERE o_totalprice > (SELECT MIN(o_totalprice)
                      FROM orders
                      WHERE o_orderpriority = '2-HIGH')

Tempo de execução = 34.640ms
```

Figura 4.17: Comando SQL Q_{15} (consulta gerada após a aplicação da heurística H6 sobre Q_{14}).

$, <=, >, >=, <>$) com os elementos de um conjunto retornado por uma subconsulta. Este operador devolve *TRUE* se pelo menos uma das linhas do conjunto satisfazem a condição, ou seja, devolve *FALSE* se nenhuma linha pertencente ao conjunto satisfaz a condição. Um exemplo da aplicação da heurística H6 é apresentado nas Figuras 4.16 e 4.17. O comando SQL Q_{14} (Figura 4.16) seleciona todas as linhas da tabela "*orders*" que possuem o preço total ("*o_totalprice*") menor do que pelo menos um dos elementos retornado por sua subconsulta. Note que o operador *SOME* pode ser removido caso a função *MIN* seja inserida na subconsulta. Assim, ao invés de comparar "*o_totalprice*" como todos os elementos retornados pela subconsulta, "*o_totalprice*" seria comparado com apenas um elemento, uma vez que agora a subconsulta retorna apenas um elemento (o valor mínimo de *o_totalprice*). O comando SQL Q_{15} (Figura 4.17) é gerado pela aplicação da heurística H6 no comando SQL Q_{14} (Figura 4.16). A heurística H6 foi identificada a partir de (ELMASRI; NAVATHE, 2005).

```
SELECT o_orderkey, o_custkey, o_orderstatus, o_totalprice
FROM orders
WHERE o_totalprice > ANY (SELECT o_totalprice
                          FROM orders
                          WHERE o_orderpriority = '2-HIGH')

Tempo de execução = 50.341ms
```

Figura 4.18: Comando Q_{16} (consulta contendo um operador sobre conjuntos *ANY* desnecessário).

```
SELECT o_orderkey, o_custkey, o_orderstatus, o_totalprice
FROM orders
WHERE o_totalprice > (SELECT MIN(o_totalprice)
                      FROM orders
                      WHERE o_orderpriority = '2-HIGH')

Tempo de execução = 31.382ms
```

Figura 4.19: Comando Q_{17} (consulta gerada após a aplicação da heurística H7 sobre Q_{16}).

4.1.7 Heurística H7

A heurística H7 busca remover o operador sobre conjuntos *ANY* (sinônimo de *SOME*) quando este é desnecessário. O operador *ANY* permite a uma consulta externa fazer comparações usando relacionais ($=, <, <=, >, >=, <>$) com os elementos de um conjunto retornado por uma subconsulta. Este operador devolve *TRUE* se pelo menos uma das linhas do conjunto satisfazem a condição, ou seja, devolve *FALSE* se nenhuma linha pertencente ao conjunto satisfaz a condição. Um exemplo da aplicação da heurística H7 é apresentado nas Figuras 4.18 e 4.19. O comando SQL Q_{16} (Figura 4.18) seleciona todas as linhas da tabela "*orders*" que possuem o preço total ("*o_totalprice*") menor do que pelo menos um dos elementos retornado por sua subconsulta. Note que o operador *ANY* pode ser removido caso a função *MIN* seja inserida na subconsulta. Assim, ao invés de comparar "*o_totalprice*" como todos os elementos retornados pela subconsulta, "*o_totalprice*" seria comparado com apenas um elemento, uma vez que agora a subconsulta retorna apenas um elemento (o valor mínimo de *o_totalprice*). O comando SQL Q_{17} (Figura 4.19) é gerado pela aplicação da heurística H7 no comando SQL Q_{16} (Figura 4.18). A heurística H7 foi identificada a partir de (ELMASRI; NAVATHE, 2005).

4.1.8 Heurística H8

A heurística H8 procura trocar o operador sobre conjuntos *IN* por uma operação de junção.

O operador *IN* permite a uma consulta externa verificar se um determinado valor pertence a um conjunto retornado por uma subconsulta. Este operador devolve *TRUE* se o valor avaliado

```
SELECT o_orderkey,o_custkey
FROM orders
WHERE o_orderkey IN (SELECT l_orderkey
                     FROM lineitem

Tempo de execução = 62.354ms
```

Figura 4.20: Comando SQL Q_{18} (consulta que utiliza o operador sobre conjuntos *IN*).

```
SELECT o_orderkey,o_custkey
FROM orders, lineitem
WHERE (o_orderkey = l_orderkey)

Tempo de execução = 35.097ms
```

Figura 4.21: Comando SQL Q_{19} (consulta gerada a partir de Q_{18}, após a aplicação da heurística H8).

pertence ao conjunto gerado pela subconsulta e retorna *FALSE* caso contrário. Um exemplo da aplicação da heurística H8 é apresentado nas Figuras 4.20 e 4.21. O comando SQL Q_{18} (Figura 4.20) seleciona todas as linhas da tabela "*orders*" cujo valor para o atributo ("*o_orderkey*") pertença ao conjunto de valores retornados pela subconsulta. Observe que o operador *IN* pode ser substituído por uma operação de junção. O comando SQL Q_{19} (Figura 4.21) é gerado pela aplicação da heurística H8 ao comando SQL Q_{18} (Figura 4.20). A heurística H8 foi identificada a partir de (ELMASRI; NAVATHE, 2005; SHASHA; BONNET, 2003; RAMAKRISHNAN; GEHRKE, 2008).

4.1.9 Heurística H9

A heurística H9 busca remover o operador *DISTINCT* quando este é desnecessário. Esse fato ocorre quando o operador *DISTINCT* é aplicado a um atributo definido como chave primária da relação e uma das seguintes situações é verificada:

- A instrução SQL possui apenas uma tabela.
- A instrução SQL possui mais de uma tabela e o atributo definido como chave primária (sobre o qual é aplicado o operador *DISTINCT*) não é chave estrangeira em nenhuma das demais tabelas presentes na cláusula *FROM* do comando SQL.

Nestas duas situações, o uso do operador *DISTINCT* degrada o desempenho do comando SQL uma vez que implica na execução de uma operação de ordenação adicional para eliminar as tuplas com dados repetidos (o que envolve elevados custos tanto computacional quanto de I/O).

```
SELECT DISTINCT(o_orderkey), o_totalprice, c_name
FROM customer, orders
WHERE c_custkey = o_custkey
ORDER BY o_orderkey

Tempo de execução = 52.634ms
```

Figura 4.22: Comando SQL Q_{20} (consulta contendo um operador *DISTINCT* desnecessário).

```
SELECT o_orderkey, o_totalprice, c_name
FROM customer, orders
WHERE c_custkey = o_custkey
ORDER BY o_orderkey

Tempo de execução = 52.634ms
```

Figura 4.23: Comando SQL Q_{21} (consulta gerada a partir de Q_{20}, após a aplicação da heurística H9).

Contudo, alguns SGBDs podem implementar o comando *DISTINCT* utilizando uma *hash*, o que também pode tornar a consulta mais lenta, já que é uma operação adicional que pode ser evitada. Por exemplo, o Oracle e o SQL Server implementa o *DISTINCT* utilizando *hash*, enquanto que o PostgreSQL utiliza ordenação.

Um exemplo da aplicação da heurística H9 é apresentado nas Figuras 4.22 e 4.23. Suponha que o atributo (coluna) *"o_orderkey"* foi definido como chave primária da tabela *orders*. Neste caso, a Figura 4.22) ilustra um comando SQL, denominado Q_{20}, contendo um operador *DISTINCT* desnecessário, pois esse operador é aplicado à coluna "*o_orderkey*", que é a chave primária da tabela "*orders*" e não é chave estrangeira na tabela ("*customer*"). A Figura 4.23 mostra o comando SQL Q_{21} gerado após a aplicação da heurística H9 sobre o comando SQL Q_{20}. A heurística H9 foi identificada em (ELMASRI; NAVATHE, 2005; SHASHA; BONNET, 2003; RAMAKRISHNAN; GEHRKE, 2008).

4.1.10 Heurística H10

Em alguns SGBDs, a ocorrência de funções aplicadas sobre colunas para as quais existem estruturas de índices associadas induz o otimizador do SGBD a desconsiderar (ou seja, não utilizar) esses índices. A heurística H10 procura identificar essas situações e modificar a expressão em que a função é utilizada, de forma que esta não mais seja aplicada sobre a coluna que possui índices associados.

Um exemplo da aplicação da heurística H10 é apresentado nas Figuras 4.24 e 4.25. Suponha a existência de uma determinada estrutura de índices *i* definida sobre o atributo "*l_quantity*" (chave de busca do índice *i*) da tabela "*lineitem*". Observe que o comando SQL Q_{22} (Figura

```
SELECT *
FROM lineitem
WHERE CAST(l_quantity AS VARCHAR) = '28'

Tempo de execução = 15.913ms
```

Figura 4.24: Comando SQL Q_{22} (consulta que utiliza uma função aplicada a uma coluna com índice).

```
SELECT *
FROM lineitem
WHERE l_quantity = CAST('28' AS INT)

Tempo de execução = 11.580ms
```

Figura 4.25: Comando SQL Q_{23} (consulta gerada a partir de Q_{22}, após a aplicação da heurística H10).

4.24) contém uma função *CAST*, que é responsável pela conversão de tipos de dados, aplicada sobre a coluna "*l_quantity*", que corresponde à chave de busca do índice *i*. Neste caso, em geral, os otimizadores dos SGBDs geram planos de execução que não fazem uso do índice *i*. A Figura 4.25 mostra o comando SQL Q_{23} gerado após a aplicação da heurística H10 sobre o comando SQL Q_{22}. Note que a função *CAST* não é mais aplicada sobre o atributo "*l_quantity*". Isso possibilita que o otimizador gere um plano de execução que utilize o índice *i* (definido sobre "*l_quantity*"). A heurística H10 foi identificada a partir de (ELMASRI; NAVATHE, 2005; SHASHA; BONNET, 2003).

4.1.11 Heurística H11

Em alguns SGBDs, a ocorrência de expressões aritméticas envolvendo colunas para as quais existem estruturas de índices associadas induz o otimizador do SGBD a desconsiderar (ou seja, não utilizar) esses índices. A heurística H11 procura identificar essas situações e modificar a expressão aritmética, de forma que esta não mais envolva a coluna que possui índices associados.

Um exemplo da aplicação da heurística H11 é apresentado nas Figuras 4.26 e 4.27. Suponha a existência de uma determinada estrutura de índices *i* definida sobre o atributo "*l_quantity*" (chave de busca do índice *i*) da tabela "*lineitem*". Observe que o comando SQL Q_{24} (Figura 4.24) contém uma expressão aritmética que envolve o atributo "*l_quantity*" ("*l_quantity*"/2=40). Neste caso, em geral, os otimizadores dos SGBDs geram planos de execução que não fazem uso do índice *i*. A Figura 4.27 mostra o comando SQL Q_{25} gerado após a aplicação da heurística H11 sobre o comando SQL Q_{24}. Note que a expressão aritmética "*l_quantity*"/2=40 foi substituída pela expressão "*l_quantity*"=40*2. Assim, o atributo "*l_quantity*" ficou isolado em um

```
SELECT *
FROM lineitem
WHERE l_quantity/2 = 40

Tempo de execução = 3.638ms
```

Figura 4.26: Comando SQL Q_{24} (consulta contendo uma expressão aritmética envolvendo uma coluna com índice).

```
SELECT *
FROM lineitem
WHERE l_quantity = 40*2

Tempo de execução = 14ms
```

Figura 4.27: Comando SQL Q_{25} (consulta gerada a partir de Q_{24}, após a aplicação da heurística H11).

dos lados da expressão aritmética. Isso possibilita que o otimizador gere um plano de execução que utilize o índice *i* (definido sobre "*l_quantity*"). A heurística H11 foi identificada em (ELMASRI; NAVATHE, 2005; RAMAKRISHNAN; GEHRKE, 2008).

4.2 Classificação das Heurísticas para Sintonia de Instruções SQL

Nessa Seção, propomos uma classificação para as heurísticas de reescrita de instruções SQL apresentada na Tabela 4.1. A classificação proposta baseia-se nas operações e estruturas utilizadas nas instruções SQL. Neste sentido, as heurísticas foram organizadas em quatro grupos: heurísticas baseadas em operações sobre conjuntos, heurísticas baseadas em operações de ordenação, heurísticas baseadas na utilização de estruturas de índice e heurísticas baseadas na utilização de tabelas temporárias.

As heurísticas baseadas em operações sobre conjuntos procuram reescrever instruções SQL diminuindo a quantidade de comparações feitas pelos operadores sobre conjuntos (ALL, SOME, ANY e IN) com suas subconsultas. Neste grupo se enquadram as heurísticas H5, H6, H7 e H8. Quanto menos comparações forem feitas nas instruções, menor será o tempo gasto para executá-las.

As heurísticas baseadas em operações de ordenação ou *hash* tentam reescrever instruções SQL que envolvem operações de ordenação desnecessárias. As heurísticas H2, H3 e H9 encontram-se neste grupo. As ordenações são operações bastante custosas ao SGBD devido a elevada taxa de I/O. Contudo, às vezes os SGBDs utilizam operações de *hash* ao invés de ordenação nessas

heurísticas.

As heurísticas baseadas na utilização de estruturas de índice procuram reescrever instruções SQL que não fazem uso dos índices disponíveis de maneira que essas estruturas passem a ser utilizadas. Neste grupo encontram-se as heurísticas H5, H10 e H11. Os índices são estruturas de acesso em bancos de dados que otimizam a busca de dados, permitindo uma localização mais rápida de um registro.

Por fim, as heurísticas baseadas na utilização de tabelas temporárias têm por objetivo eliminar a criação de tabelas temporárias. A heurística H1 compõe este grupo. A criação física tabelas temporárias consome bastantes recursos do sistema de bancos de dados.

4.3 Implementação das Heurísticas para Sintonia de Instruções SQL pelos SGBDs Comerciais

A Seção 4.1 descreve um conjunto de heurísticas para a reescrita de instruções SQL que foram identificadas a partir de uma extensa pesquisa bibliográfica em (ELMASRI; NAVATHE, 2005; SHASHA; BONNET, 2003; RAMAKRISHNAN; GEHRKE, 2008). Contudo, não encontramos na literatura nenhum trabalho que avaliasse a utilização dessas heurísticas pelos SGBDs comerciais.

Por este motivo, realizamos um experimento com a finalidade de verificar, para cada uma das heurísticas previamente identificadas (H1 a H11), quais destas heurísticas para reescrita de instruções SQL já são implementadas (aplicadas) pelos principais SGBDs comerciais (PostgreSQL 8.1, SQL Server 2008 e Oracle 10g). Neste sentido, elaboramos, a partir da base dados do Benchmark TPC-H, três instruções SQL para cada uma das heurísticas identificadas (H1 a H11), totalizando 33 instruções SQL.

Assim, para a heurística H1 elaboramos três instruções SQL onde esta heurística pudesse ser aplicada: Q_1, Q_2 e Q_3. Em seguida, reescrevemos cada uma dessas três instruções utilizando a heurística H1. Desta forma, foram geradas as instruções Q'_1, Q2'e Q3'. Onde Q1' corresponde à instrução gerada a partir da aplicação da heurística H1 sobre a instrução Q1. Logo, formamos três pares de instruções SQL: (Q1, Q1'), (Q2, Q2') e (Q3, Q3'). Cada par de instruções SQL agrupa uma instrução SQL original (Q1, por exemplo) e a instrução SQL produzida após a aplicação da heurística H1 (Q1', por exemplo) sobre a instrução inicial.

Para verificar se a heurística H1 já é implementada (utilizada) por um determinado SGBD (PostgreSQL, por exemplo), executamos cada par de instruções SQL e verificamos se os planos de execução gerados para a instrução SQL original e para a instrução SQL reescrita são iguais. Caso o resultado desta verificação seja verdadeiro para cada um dos três pares de instruções SQL utilizados, consideramos que a heurística H1 já é implementada pelo SGBD. Neste caso, constatamos que o SGBD já reescreve a instrução SQL automaticamente, ou seja, já aplica a heurística H1 de forma automática. Caso contrário, assumimos que a heurística H1 não é implementada pelo SGBD. Esse procedimento foi utilizado para cada uma das heurísticas de rescrita de instruções SQL descritas na Seção 4.1 e repetido para os três principais SGBDs comerciais

Tabela 4.2: Heurísticas já implementadas pelos SGBDs comerciais.

Heurística	PostgreSQL	SQL Server	Oracle
H1	Não	Não	Não
H2	Não	Sim	Não
H3	Não	Não	Não
H4	Não	Não	Sim
H5	Não	Não	Sim
H6	Não	Não	Sim
H7	Não	Não	Sim
H8	Não	Não	Sim
H9	Não	Não	Não
H10	Não	Não	Não
H11	Não	Não	Não

(PostgreSQL, SQL Server e Oracle). O ambiente de experimentação utilizado foi composto por uma estação Core i3-21003.10GHz, com 4GB de Ram e 500 GB de Hd. Adicionalmente, analisamos as documentações dos fabricantes dos três SGBDs analisados buscando informações acerca das heurísticas de reescritas aplicadas automaticamente.

A Tabela 4.2 resume os resultados do experimento realizado. Assim, cada linha da tabela representa uma heurística. A primeira coluna indica o nome da heurística. As três colunas seguintes indicam se a heurística é ou não aplicada automaticamente pelos SGBDsPostgreSQL, SQL Server e Oracle, nesta ordem. Observando a Tabela 4.2 pode-se constatar que os SGBDs PostgreSQL e SQLServer não implementam nenhuma das heurísticas apresentadas nessa tabela, enquanto que o SGBD Oracle já implementa as heurísticas H4, H5, H6, H7 e H8.

4.4 Heurísticas Não Automatizáveis

Algumas heurísticas para a reescrita de instruções SQL não podem ser aplicadas de forma automática, uma vez que dependem de informações que não podem ser encontradas diretamente no texto do comando SQL. Para ilustrar esses casos considere o comando SQL Q_{26} mostrado na Figura 4.28. Assuma que o usuário gostaria de recuperar apenas os atributos *c_custkey* e *c_name*, mas escreveu *SELECT* * apenas por comodidade. Neste caso, seria mais eficiente recuperar somente as colunas desejadas. Logo, uma possível heurística, denominada H12, consistiria em substituir a expressão *SELECT* * por uma expressão que retornasse somente as colunas desejadas pelo usuário, como mostra a Figura 4.29. Contudo, não há como descobrir quais as colunas que o usuário deseja analisando apenas o texto do comando SQL. Portanto, consideramos que esta é uma heurística não automatizável. Heurísticas deste tipo não poderiam ser utilizadas nas abordagens propostas neste trabalho para a sintonia assistida e automática de instruções SQL. Contudo, o protótipo implementado pode detectar e informar ao DBA, por meio de relatórios, a existência de instruções para as quais essa heurística pudesse ser aplicada, apesar de não conseguir reescrever automaticamente o comando SQL.

```
SELECT *
FROM customer
```

Figura 4.28: Comando SQL Q_{26} (consulta que recupera todas as colunas de uma tabela).

```
SELECT c_custkey, c_name
FROM customer
```

Figura 4.29: Comando SQL Q_{27} (consulta gerada após a aplicação da heurística não automatizável H12 sobre a consulta Q_{26}).

5 ABORDAGENS PARA SINTONIA DE INSTRUÇÕES SQL

5.1 Visão Geral

Para auxiliar o DBA na complexa tarefa de ajustar comandos SQL, este trabalho propõe duas abordagens distintas para a sintonia de instruções SQL em bancos de dados relacionais: uma abordagem assistida (Sessão **??**) e outra automática (Sessão 5.3). A abordagem assistida procura envolver e interagir com o DBA, especialista em sintonia de instruções SQL, com o objetivo de tirar proveito de sua experiência e conhecimento para proporcionar bons resultados. A abordagem automática, por sua vez, busca realizar a sintonia de comandos SQL de forma completamente independente da interação com o DBA. Assim, a abordagem automática é direcionada para cenários onde um DBA não está disponível ou sua interferência não seria viável, como por exemplo, aplicações embarcadas ou disponibilizadas em plataformas de computação em nuvem.

Para cada uma das abordagens propostas nesse trabalho foi concebida uma arquitetura. De um modo geral, as arquiteturas propõem a realização de sintonia durante a operação normal do SGBD, por meio da colaboração entre agentes de *software*, com a finalidade de fornecer uma solução completa e eficiente para o problema da reescrita de instruções SQL. Esses agentes monitoram e interagem com o sistema de banco de dados, por meio de *drivers*, para obter diversas informações que são utilizadas no processo de sintonia, tais como: as consultas previamente executadas, os planos de execução gerados para essas consultas, as estruturas de índices existentes, os atributos definidos como chaves primárias e estrangeiras, a cardinalidade das relações, além de outras estatísticas. Um *driver* consiste em uma *interface* Java que contém um conjunto de métodos abstratos. Assim, para cada SGBD que se deseja utilizar torna-se necessário codificar uma classe Java que implementa a *interface* correspondente ao *driver*. Contudo, o protótipo implementado para validar as ideias propostas nesta dissertação já fornece *drivers* instanciados para os SGBDs PostgreSQL 8.1, Oracle 10g e SQL Server 2008. Os agentes, de posse dos dados recuperados a partir do SGBD, utilizam heurísticas, que encapsulam o conhecimento de especialistas em sintonia de instruções SQL. As heurísticas são constituídas de regras para identificar potenciais problemas ou oportunidades de sintonia nos comandos SQL, bem como formas de ajustar (reescrever) as instruções SQL para solucionar os problemas identificados ou explorar as oportunidades observadas. A Tabela 4.1 ilustra as onze heurísticas para reescrita de instruções SQL utilizadas nas duas abordagens propostas neste trabalho.

As duas abordagens propostas apresentam ainda as seguintes características:

- São Não-intrusivas: as abordagens assistida e automática são completamente desacoplada do código-fonte do SGBD utilizado. Isso permite que a solução concebida possa ser utilizada com qualquer SGBD.
- São Independentes de localização: as duas abordagens propostas podem executar em uma máquina distinta daquela utilizada para hospedar o SGBD, não consumindo recursos do servidor onde o SGBD está hospedado. *On-the-fly*: são executadas sempre que necessário e de forma contínua durante o funcionamento normal do SGBD.

Adicionalmente, o funcionamento da abordagem automática é completamente independente de interações com seres humanos.

5.2 Uma Abordagem para a Sintonia Assistida de Instruções SQL

A abordagem assistida (ou iterativa) consiste em um *advisor* que tem por objetivo:

1. capturar automaticamente as instruções SQL anteriormente executadas e armazená-las em uma *metabase* local; ;
2. periodicamente, analisar as instruções armazenadas na *metabase* local à procura de oportunidade de sintonia (reescrita) e
3. sugerir (por meio de alertas, *wizards* ou relatórios) oportunidades de sintonia (reescrita).

Desta forma, o *advisor* identifica instruções SQL que se fossem reescritas poderiam fazer com que o otimizador de consultas escolhesse planos de execução melhores, reduzindo o tempo de resposta dos comandos SQL. Adicionalmente, o *advisor* permite ao DBA interagir com o processo de sintonia, utilizando os princípios da estratégia de sintonia assistida ou interativa (ALAGIANNIS et al., 2010; BRUNO; CHAUDHURI; RAMAMURTHY, 2009; MAIER et al., 2010). Neste sentido, o DBA pode, por exemplo, selecionar um subconjunto das heurística disponibilizadas a fim de que somente essas sejam utilizadas para reescrever as instruções SQL em geral ou um determinado comando SQL em particular. Assim, se o DBA identificar que algumas heurísticas são desnecessárias ou inadequadas ao seu banco de dados ou para uma determinada instrução SQL, ele pode desativá-las.

A arquitetura concebida para a abordagem assistida é exibida na Figura 5.1. Os principais componentes desta arquitetura são:

- **Agent for Workload Obtainment (AWO)**: o agente *AWO* é responsável por consultar a metabase do SGBD, recuperar as instruções SQL que estão sendo executadas e armazená-las na *metabase* local (*Local MetaData*).

- **Driver for Workload Access (DWA)**: esse *driver* permite ao agente *AWO* recuperar a carga de trabalho submetida a um SGBD específico.
- **Local MetaData (LM)**: a *LM* armazena a carga de trabalho capturada pelo agente *AWO*.
- **Agent for SQL Tuning (AST)**: realiza a sintonia de uma determinada instrução SQL utilizando o conjunto de heurísticas (*HS*).
- **Heuristic Set (HS)**: conjunto de heurísticas utilizadas pelo agente *AST* para identificar instruções SQL com oportunidades de sintonia e reescrever essas instruções. Inicialmente, implementamos 11 heurísticas. Contudo, novas heurísticas podem ser implementadas e adicionadas à arquitetura proposta. As 11 heurísticas implementadas são apresentadas na Seção 4.1.
- **Agent for Statistics Obtainment (ASO)**: algumas heurísticas necessitam de informações estatísticas (como, por exemplo, as chaves primárias de uma determinada tabela, as estruturas de índice existentes, etc.). O agente *ASO* é responsável por acessar o SGBD e recupera essas informações estatísticas.
- **Driver for Statistics Access (DSA)**: *driver* que permite ao agente *ASO* recuperar as estatísticas de um SGBD específico.
- **Driver for Data Access (DDA)**: Esse *driver* permite que o motor do *middleware* proposto (*MST*) envie as instruções SQL ajustadas para o SGBD e receba o resultado da execução dessas instruções.
- **Tuning Settings**: funciona como um repositório de configurações de sintonia de instruções SQL. Esse repositório armazena *tuplas* com o seguinte formato: <subconjunto de heurísticas, instrução SQL>. Assim, cada*tuplas* especifica as heurísticas que devem ser aplicadas na reescrita de uma determinada instrução S L.

5.3 Uma Abordagem para a Sintonia Automática de Instruções SQL

A abordagem automática consiste em um *middleware* que atua entre a aplicação e o SGBD. Este *middleware* é responsável por:

1. receber as instruções SQL enviadas pelas aplicações;
2. analisar e reescrever as instruções SQL recebidas (se necessário);
3. enviar as instruções (reescritas ou não) para o SGBD;
4. receber do SGBD o resultado da execução de cada instrução SQL e

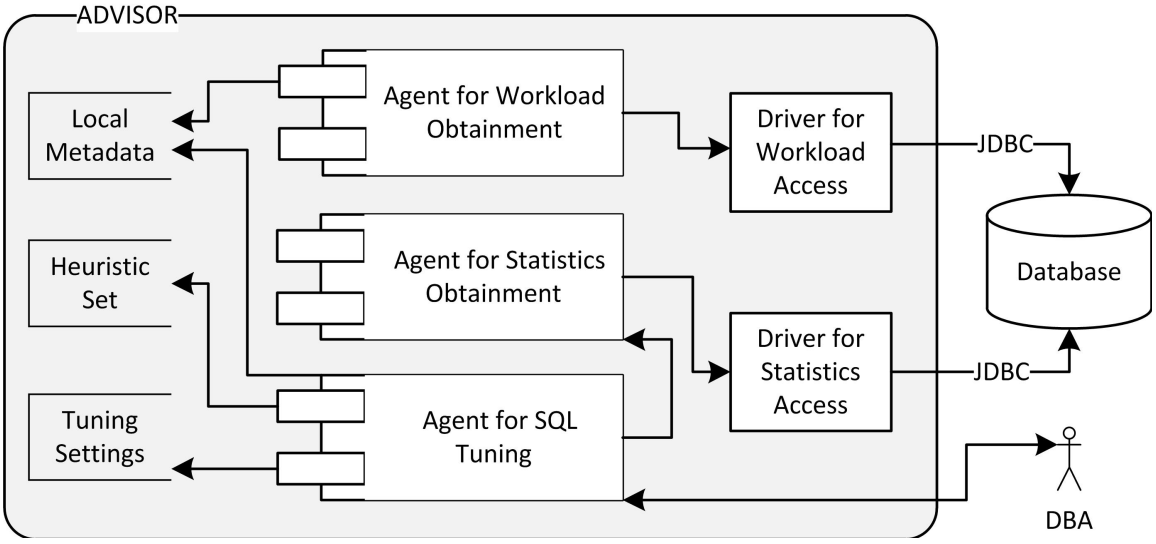

Figura 5.1: Arquitetura para sintonia assistida de instruções SQL.

5. enviar o resultado da execução de cada comando SQL para a aplicação cliente.

A abordagem automática pode ainda utilizar as preferências definidas pelo DBA por meio *advisor*, tanto para comandos SQL em geral quanto para uma instrução SQL em particular. Além disso, opcionalmente, o DBA pode configurar o funcionamento do *midlleware* para que este verifique, antes de enviar um determinado comando SQL Q' (gerado durante o processo de reescrita da instrução SQL Q) para execução, se o custo de execução estimado de Q' (instrução reescrita) pelo SGBD é menor do que o custo de execução estimado de Q (instrução SQL original), com a finalidade de assegurar que o comando gerado pelo processo de reescrita Q' só será utiliza caso efetivamente proporcione ganhos de desempenho. Logicamente, o sucesso dessa estratégia depende da atualização das estatísticas do SGBD. Além disso, essa verificação implica na execução de consultas adicionais à *metabase* do SGBD, o que diminui o desempenho do processo de reescrita.

A arquitetura concebida para a abordagem automática é ilustrada na Figura 5.2. O principal componente desta arquitetura, além dos componentes que também estão presentes na arquitetura da abordagem assistida (Figura 5.1), e que já foram anteriormente comentados, é o MST (*Middleware for SQL Tuning*), o qual discutido a seguir:

- **Middleware for SQL Tuning (MST)**: é o módulo responsável por receber instruções SQL das aplicações, enviá-las ao agente *MST* para que ele as analise e receber as instruções ajustadas (ou não). Em seguida, o *MST* envia as instruções SQL para o SGBD, receber o resultado da execução dessas instruções e as envia para as aplicações de origem.

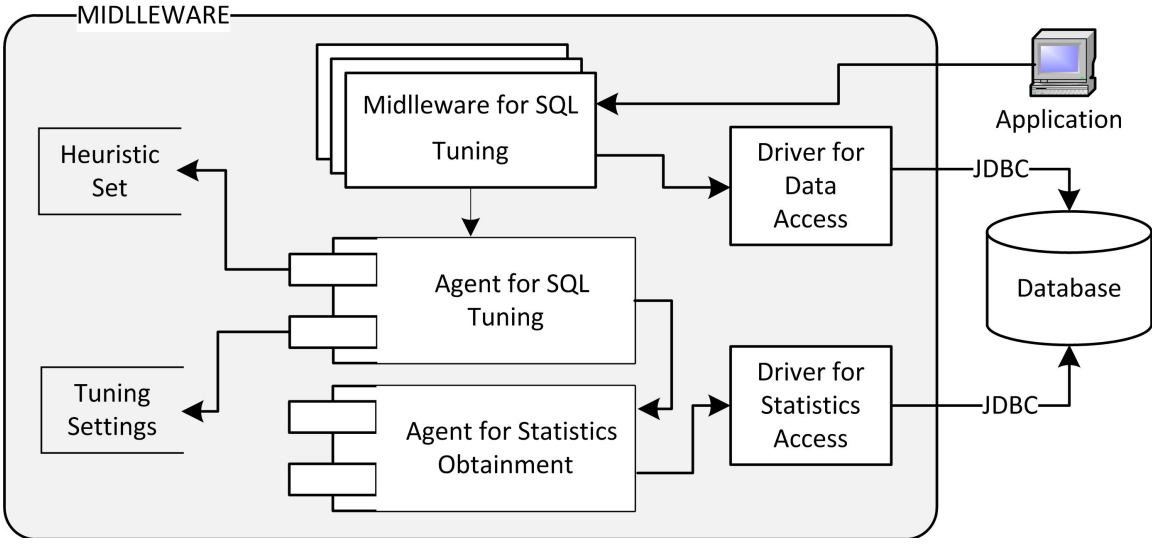

Figura 5.2: Arquitetura para sintonia automática de instruções SQL.

5.4 Detalhes de Implementação

Dada uma determinada instrução SQL Q, o processo de reescrita, por meio das heurísticas apresentadas na Tabela 3.1, deve percorrer o texto de Q na busca de problemas ou oportunidades de sintonia (como por exemplo, a existência de um operador *DISTINCT* desnecessário) e, posteriormente, alterar a sintaxe de Q, gerando um novo comando Q'. Para isso, utilizamos um *parser* para instruções SQL denominado *ZQL* (THURAISINGHAM et al., 2010). O parser *ZQL* é escrito em linguagem Java e foi implementado a partir do JavaCC (*Java Compiler Compiler*) (KODAGANALLUR, 2004). O JavaCC é um gerador de *parser* para aplicações Java. Um gerador de *parser* é uma ferramenta que lê as especificações de uma gramática e as converte em um programa capaz de reconhecer instruções construídas com essa gramática. O *parser* ZQL foi desenvolvido com JavaCC utilizando uma gramática para instruções SQL. Além de reconhecer instruções SQL, o *ZQL* armazena os elementos de uma instrução SQL em estruturas de dados Java. Assim, para um determinado comando SQL são gerados objetos que armazenam a lista de colunas, a lista de tabelas, as expressões aritméticas ou lógicas, dentre outras. Esses objetos foram utilizados na implementação das heurísticas de sintonia de instruções SQL.

Alguns outros *parsers* para instruções SQL também foram investigados e analisados, como, por exemplo: o *General SQL Parser* (MERLO; LETARTE; ANTONIOL, 2007) e o *SQL Query Parser* (ELLIOTT et al., 2009). Entretanto, o *parser ZQL* foi escolhido por ser uma ferramenta de fácil utilização, livre (*open-sourse*) e que permite a inclusão de novas regras em sua gramática.

As arquiteturas para sintonia de instruções SQL propostas nesse trabalho possuem praticamente os mesmos componentes. Contudo, cada arquitetura funciona de forma diferente e bem definida. A seguir, iremos descrever, em maiores detalhes, como os componentes de cada abordagem trabalham em conjunto para solucionar o problema da reescrita de instruções SQL. As figuras 5.3 e 5.4 detalham o funcionamento da abordagem assistida, o qual pode ser descrito a partir dos seguintes passos:

1. O Agente *AWO* coleta, periodicamente, as instruções SQL executadas pelo SGBD, por meio do *driver DWA*.

2. As instruções coletadas pelo agente *AWO* são armazena na *Local Metadata*.

3. Periodicamente, o agente *AST* analisa as instruções armazenadas na *Local Metadata*. Assim, para cada instrução SQL capturada Q, o egente *AST* utiliza o parser *ZQL* para transformar o texto da instrução SQL em um objeto Java.

4. O agente *AST* utiliza as heurísticas disponíveis no *Heuristic Set* para ajustar a instrução SQL Q. Cada uma das heurísticas analisa o objeto Java criado pelo *ZQL* para representar a instrução SQL Q e, se identificar uma oportunidade de sintonia, manipula os dados do objeto de forma que a instrução que o objeto representa seja reescrita gerando uma outra instrução SQL Q' equivalente à instrução original Q.

5. Algumas das heurísticas podem precisar de informações estatísticas (tais como, verificar a presença de índices, encontrar a chave-primária de uma determinada tabela, ...), etc. Então, o agente *AST* pode requisitar essas informações ao agente *ASO* , o qual acessará o SGBD utilizando o *driver DSA*.

6. Durante o processo de sintonia da instrução SQL Q, o agente *AST* verifica se Q possui subconsultas. Em caso afirmativo, o processo de sintonia é aplicado recursivamente. O processo de sintonia procura por subconsultas nas cláusulas *SELECT*, *FROM*, *WHERE* e *HAVING*. O diagrama de atividades da Figura 5.5 e o pseudo-código da Figura 5.6 sintetiza o esquema da execução das heurísticas por meio de chamadas recursivas. Ao terminar o processo de sintonia da instrução SQL Q, por completo, o *advisor* gera um relatório contendo as recomendações de sintonia para Q. Esse relatório contém: a instrução SQL original Q e seu plano de execução estimado pelo SGBD, a instrução SQL reescrita Q' e seu plano de execução estimado pelo SGBD, além das heurísticas que foram efetivamente utilizadas na reescrita e do tempo gasto no processo de reescrita.

7. Adicionalmente, o DBA tem a possibilidade de interagir com as sugestões de sintonia fornecidas pelo *advisor*. Por exemplo, o DBA pode aplicar somente um subconjunto das heurísticas utilizadas pelo *advisor*, o DBA pode forçar a aplicação de heurísticas que não foram efetivamente utilizadas pelo *advisor*, dentre outras opções. Adicionalmente, o DBA pode escolher e definir (fixar) um subconjunto das heurísticas disponíveis para que estas sejam sempre executadas no processo de sintonia das instruções em geral ou de uma determinada instrução SQL em particular. Neste caso, essas informações sào armazenadas no repositório *Tuning Settings*.

As figuras 5.7 e 5.8 detalham o funcionamento da abordagem automática, o qual pode ser descrito a partir dos seguintes passos:

1. A aplicação envia uma instrução SQL Q para o componente *Midlleware for SQL Tuning*, a fim de que o comando Q seja reescrito.

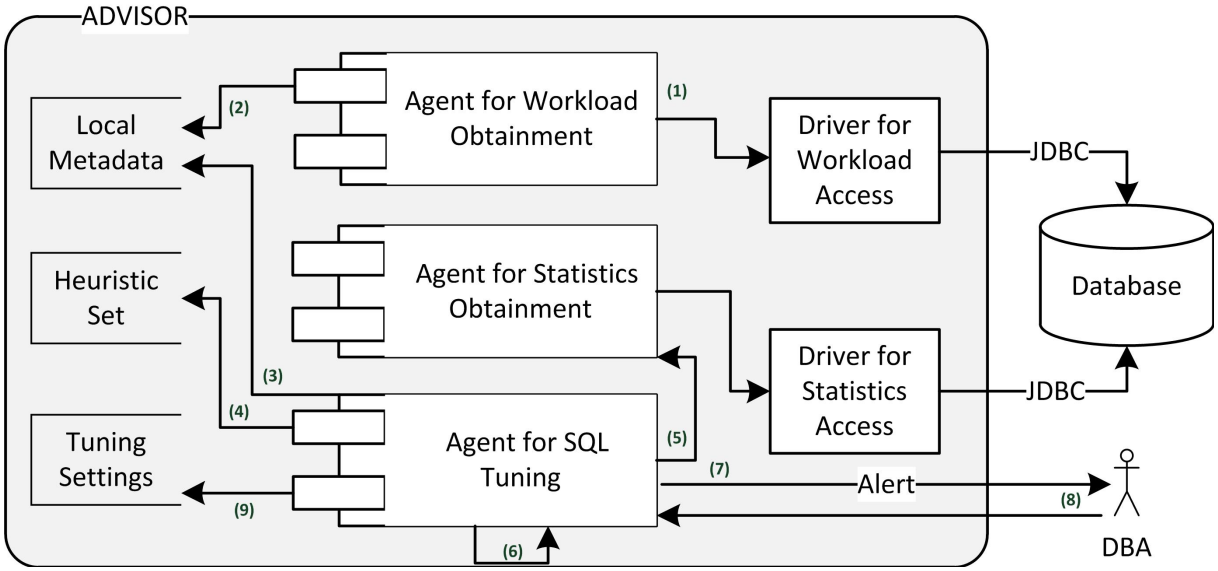

Figura 5.3: Funcionamento do *advisor* para sintonia assistida de instruções SQL.

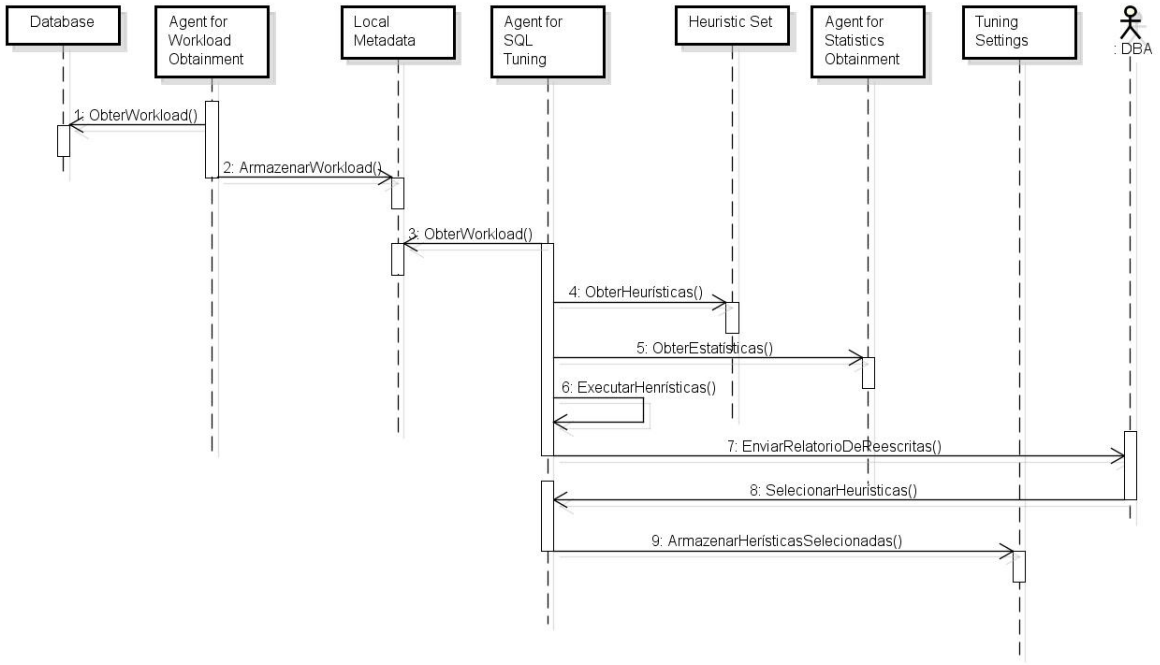

Figura 5.4: Diagrama de sequência da execução do *advisor* para sintonia assistida de instruções SQL.

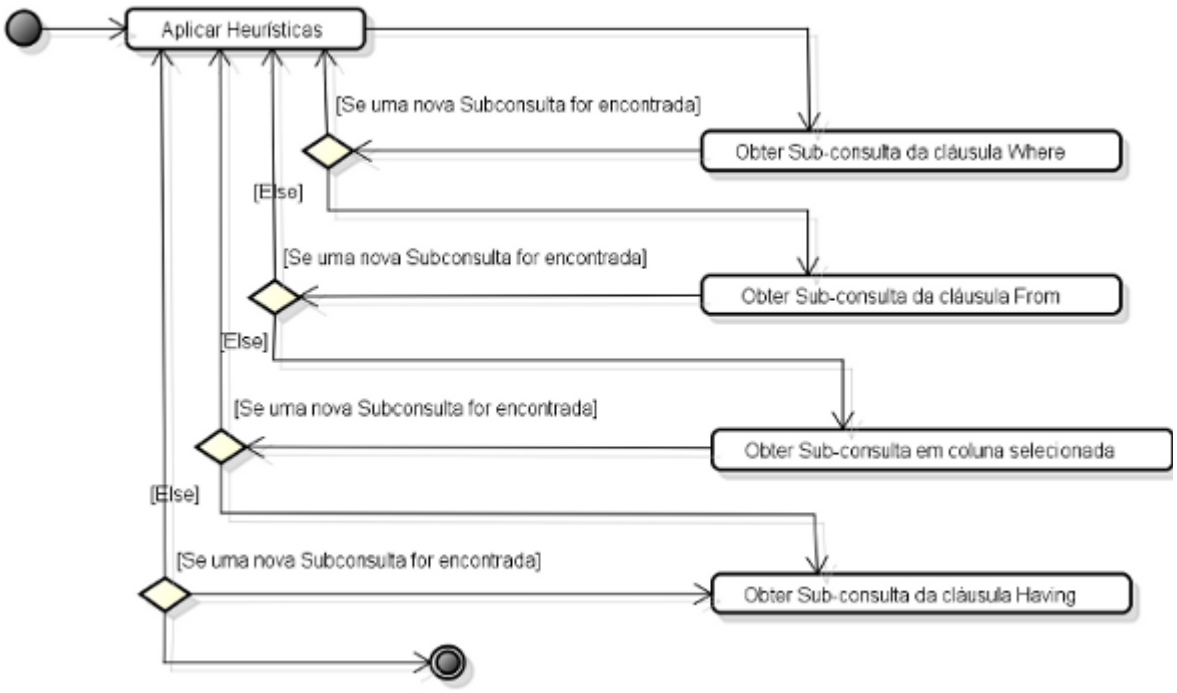

Figura 5.5: Diagrama de atividades da execução recursiva das heurísticas para sintonia de instruções SQL.

```
Função executarHeurísticas(sql)
    Para cada heurística hi do conjunto HS faça
        executar hi(sql)
    Fim Para
    Enquanto existir uma subconsulta subSql no WHERE de sql faça
        executarHeurísticas(subSql)
    Fim Enquanto
    Enquanto existir uma subconsulta subSql no FROM de sql faça
        executarHeurísticas(subSql)
    Fim Enquanto
    Enquanto existir uma subconsulta subSql em uma coluna selecionada de sql faça
        executarHeurísticas(subSql)
    Fim Enquanto
    Enquanto existir uma subconsulta subSql no HAVING de sql faça
        executarHeurísticas(subSql)
    Fim Enquanto
Fim Função
```

Figura 5.6: Algoritmo da execução recursiva das heurísticas para sintonia de instruções SQL.

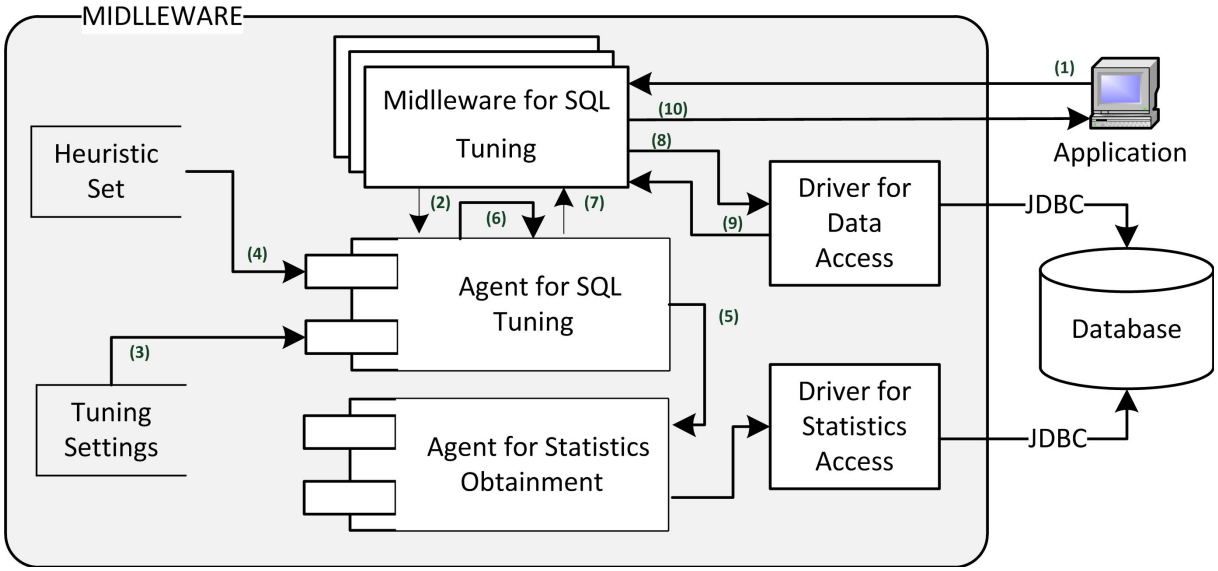

Figura 5.7: Funcionamento do *middleware* para sintonia automática de instruções SQL.

2. O *Midlleware for SQL Tuning* envia a instrução Q para o agente *AST*.

3. De posse da instrução SQL Q, o agente *AST* utiliza o *parser ZQL* para transformar o texto do comando Q em um objeto Java. Além disso, o agente *AST* verifica no componente *Tuning Settings* se a instrução Q possui um conjunto de heurísticas definidas para a sua sintonia. Caso não existam essas definições de sintonia, todas as heurísticas disponíveis serão utilizadas para ajustar a instrução Q.

4. O agente *AST* utiliza as heurísticas disponíveis no *Heuristic Set* para ajustar a instrução SQL Q. Assim, o agente *AST* aplicada cada uma das heurísticas ao comando Q. Para isso, objeto Java criado para representar a instrução SQL Q é analisado e, caso alguma possibilidade de sintonia seja identificada, a heurística correspondente é aplicada, gerando uma nova instrução Q'.

5. Algumas das heurísticas podem precisar de informações estatísticas (tais como, verificar a presença de índices, encontrar a chave-primária de uma determinada tabela, ...). Assim, o agente *AST* pode requisitar essas informações ao agente *ASO*, que acessará o SGBD utilizando o *driver DSA*.

6. O agente *AST* envia a instrução SQL Q' (caso nenhuma heurística tenha sido efetivamente aplicada, então $Q' = Q$) ao *Midlleware for SQL Tuning*.

7. De posse da instrução SQL Q', o *Midlleware for SQL Tuning* envia Q ao SGBD utilizando o *driver DDA*.

8. Após processar a consulta Q', o SGBD envia o resultado ao *Midlleware for SQL Tuning*, que irá encaminhar esse resultado à aplicação cliente.

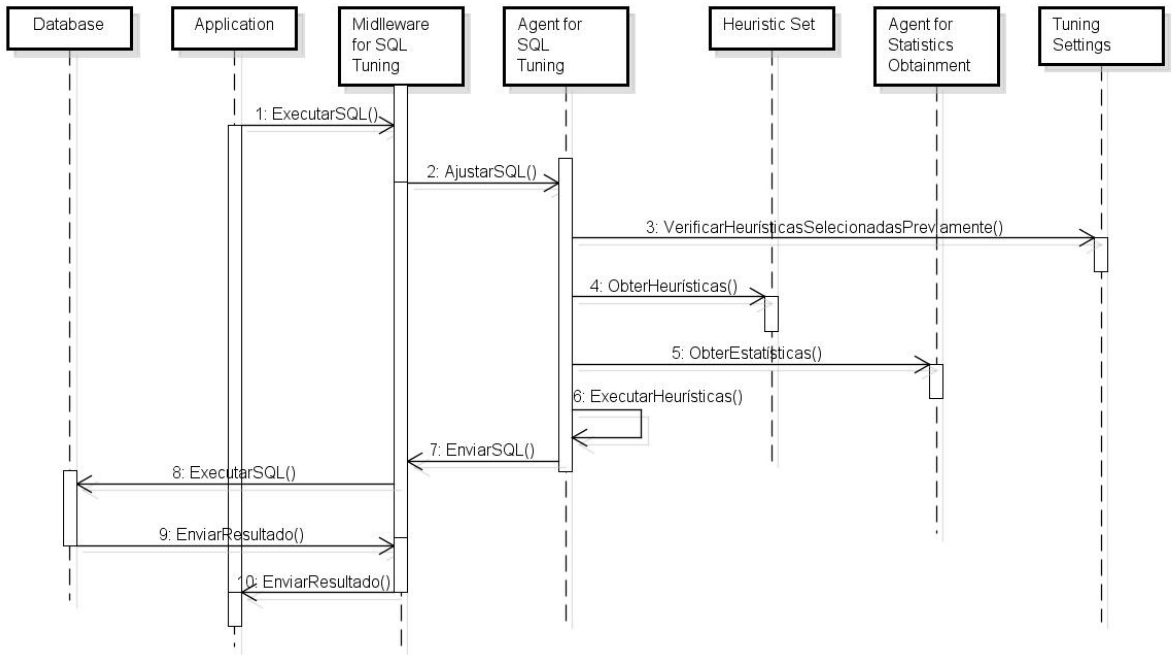

Figura 5.8: Diagrama de sequência da execução do *middleware* para sintonia automática de instruções SQL.

5.5 A Ferramenta Interactive Query Tuning (IQT)

As duas abordagens propostas (assistida e automática) permitiram a implementação de um protótipo denominado *Interactive Query Tuning* (*IQT*). O protótipo possui uma interface gráfica (Figura 5.9) que permite ao DBA interagir com o *advisor* da abordagem assistida. Adicionalmente, o *IQT* disponibiliza uma classe Java que possibilita aos desenvolvedores de aplicações utilizar o *midlleware* da abordagem automática para o ajuste automático de comandos SQL.

O *IQT* permite ao DBA interagir com o processo de sintonia de instruções SQL realizado pelo *advisor*. Contudo, para isso, é necessário fornecer à ferramenta as informações de acesso ao SGBD (tais como, endereço do servidor, usuário, senha, etc.). Essas informações devem ser fornecidas no painel de conexão como o SGBD. A tela de conexão com o SGBD é ilustrada na Figura 5.10.

Uma vez que as informações de conexão tenham sido fornecidas corretamente, o *advisor* pode ser iniciado. Para isso, basta clicar no botão *Iniciar Advisor*, o qual é ilustrado na Figura 5.11. Em execução, o *advisor* coleta e armazena, periodicamente, as instruções SQL que estão sendo executadas pelo SGBD.

O *advisor*, periodicamente, irá analisar as instruções SQL previamente armazenadas e aplicar o processo de sintonia. Caso o *advisor* consiga reescrever alguma das instruções SQL analisadas, este gera um relatório contendo as recomendações de reescritas que foram aplicadas em cada uma das instruções ajustadas. Este relatório pode ser visualizado pelo DBA sempre que este abrir a tela principal do *IQT* ou clicar no botão *Sugestões de Sintonia* na tela ilustrada na Figura 5.12. Opcionalmente, nessa mesma tela, o DBA pode visualizar também a lista de instruções

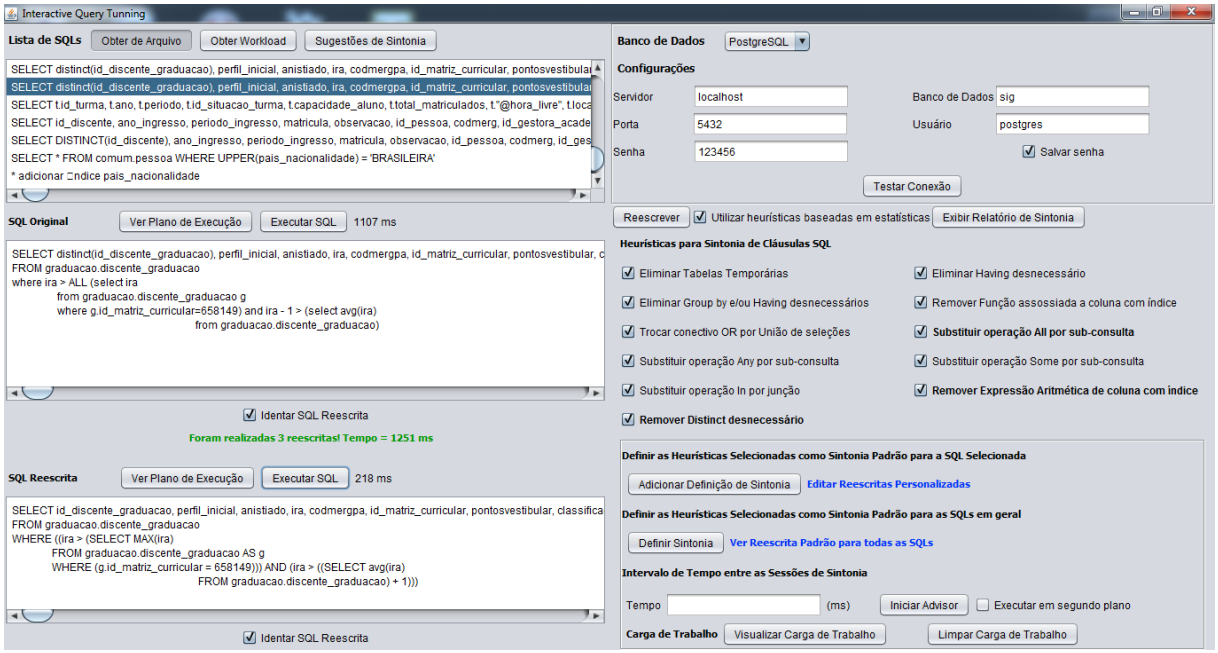

Figura 5.9: Tela Principal da Ferramenta Interactive Query Tuning.

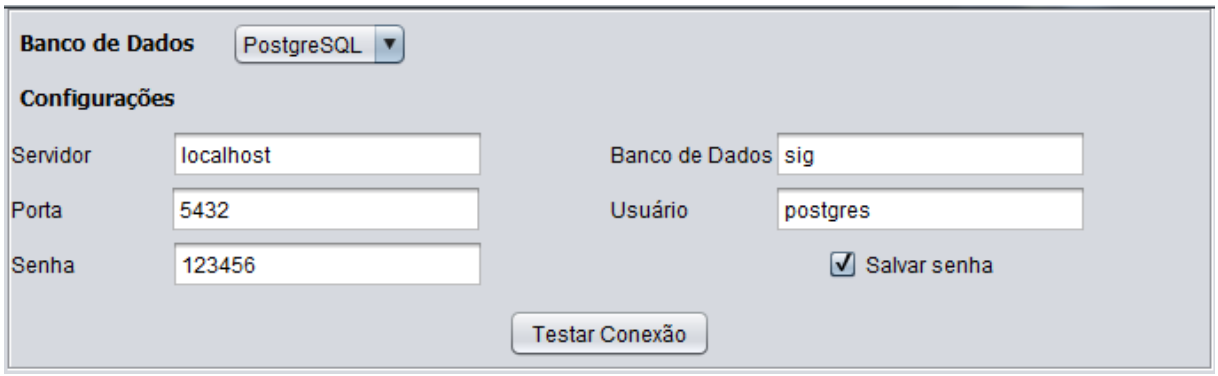

Figura 5.10: Tela de conexão com o SGBD.

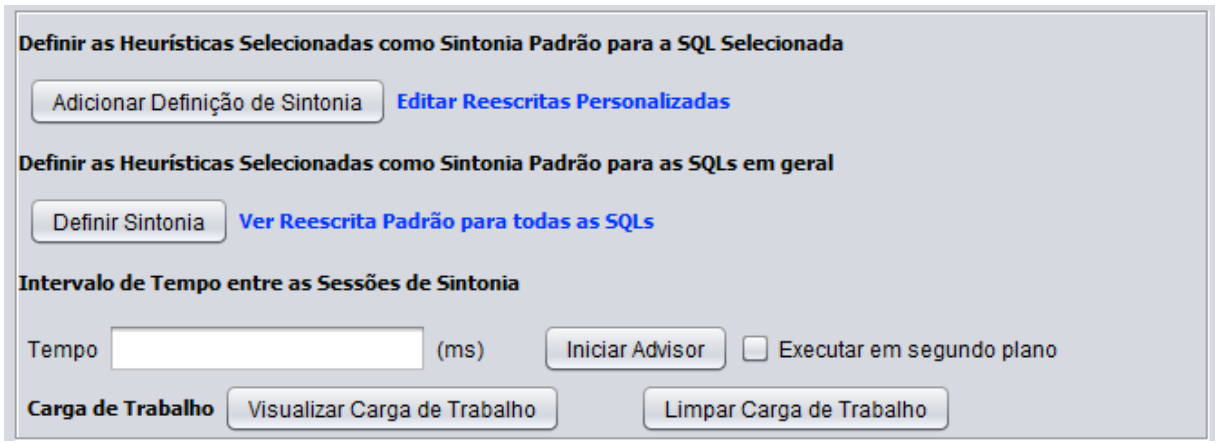

Figura 5.11: Tela de inicialização do Advisor.

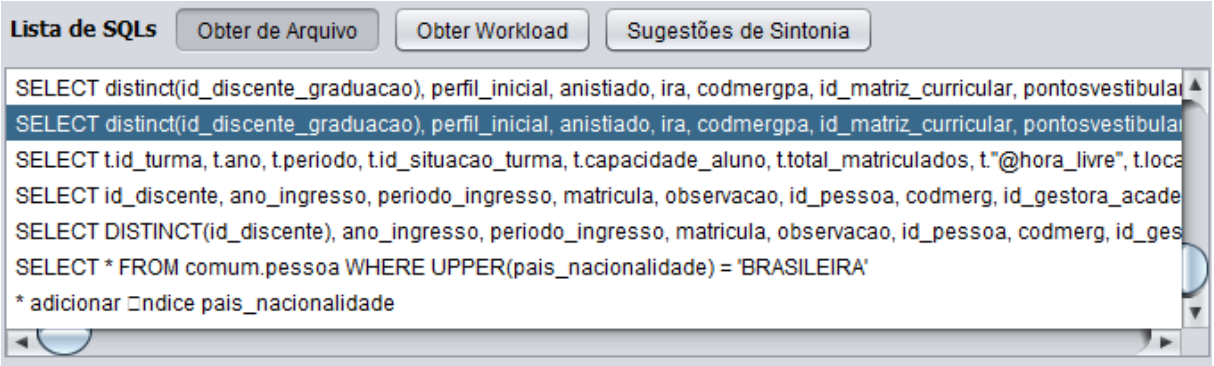

Figura 5.12: Tela de visualização da carga de trabalho e das sugestões de sintonia de instruções SQL.

SQL coletadas pelo *advisor*. Para isso, basta clicar no botão *Obter Workload*. Adicionalmente, o DBA pode fornecer um arquivo contendo um conjunto de instruções SQL a fim de que o processo de sintonia seja aplicado às instruções presentes nesse arquivo. Isso pode ser realizado clicando-se no botão *Obter Arquivo*.

O DBA pode interagir com as sugestões de sintonia geradas pelo *IQT*. Para isso, ele deve selecionar uma das instruções SQL capturas automaticamente pelo *advisor* ou fornecidas via arquivo. A instrução selecionada será exibida na caixa de texto *SQL Original*, mostrada na Figura 5.13. Para reescrever a instrução selecionada basta clicar no botão *Reescrever* exibido na Figura 5.14. Em seguida, inicia-se o processo de reescrita da instrução selecionada. Após a conclusão desse processo, caso alguma das heurísticas tenha sido aplicada com sucesso, a instrução SQL reescrita será exibida na caixa de texto *SQL Reescrita*, ilustrada na Figura 5.13.

O DBA pode ainda, por meio da ferramenta *IQT*, comparar a instrução SQL original e a instrução SQL reescrita, visualizando, por exemplo, os planos de execução e os tempos de resposta gerados para cada uma delas, bem como o tempo gasto no processo de reescrita. Para isso, basta clicar nos botões *Executar SQL* e *Ver Plano de Execução*, os quais são ilustrados na Figura 5.13. O *IQT* mostra também uma descrição de todas as transformações sofridas pela instrução SQL original durante o processo de sintonia, até a geração da consulta rescrita final. Para isso, basta clicar no botão *Exibir Relatório de Sintonia*, ilustrado na Figura 5.14.

Adicionalmente, a ferramenta *IQT* possibilita ao DBA escolher e definir quais heurísticas devem ser utilizadas no processo de reescrita de uma determinada instrução SQL. Para isso, o DBA deve selecionar a instrução SQL na *Lista de SQLs* ou deve digitar a instrução diretamente no campo *SQL Original*. Em seguida, o DBA deve selecionar as heurísticas que deseja no painel *Heurísticas para Sintonia de Cláusulas SQL*, mostrado na Figura 5.14. Nesse mesmo painel, o DBA também pode decidir por utilizar ou não heurísticas baseadas em estatísticas. Essas heurísticas são as que consomem mais tempo no processo de reescrita, uma vez necessitam fazer acessos adicionais ao SGBD para obter as informações estatísticas. Por fim, o DBA deve clicar no botão *Adicionar Definição de Sintonia* (Figura 5.11). Além disso, o DBA pode definir um subconjunto de heurísticas a serem aplicadas às instruções SQL em geral, ou seja, para as instruções SQL que não possuam definições específicas. Isso pode ser feito clicando no botão *Definir Sintonia*. Vale destacar que tanto as configurações definidas para uma instrução SQL

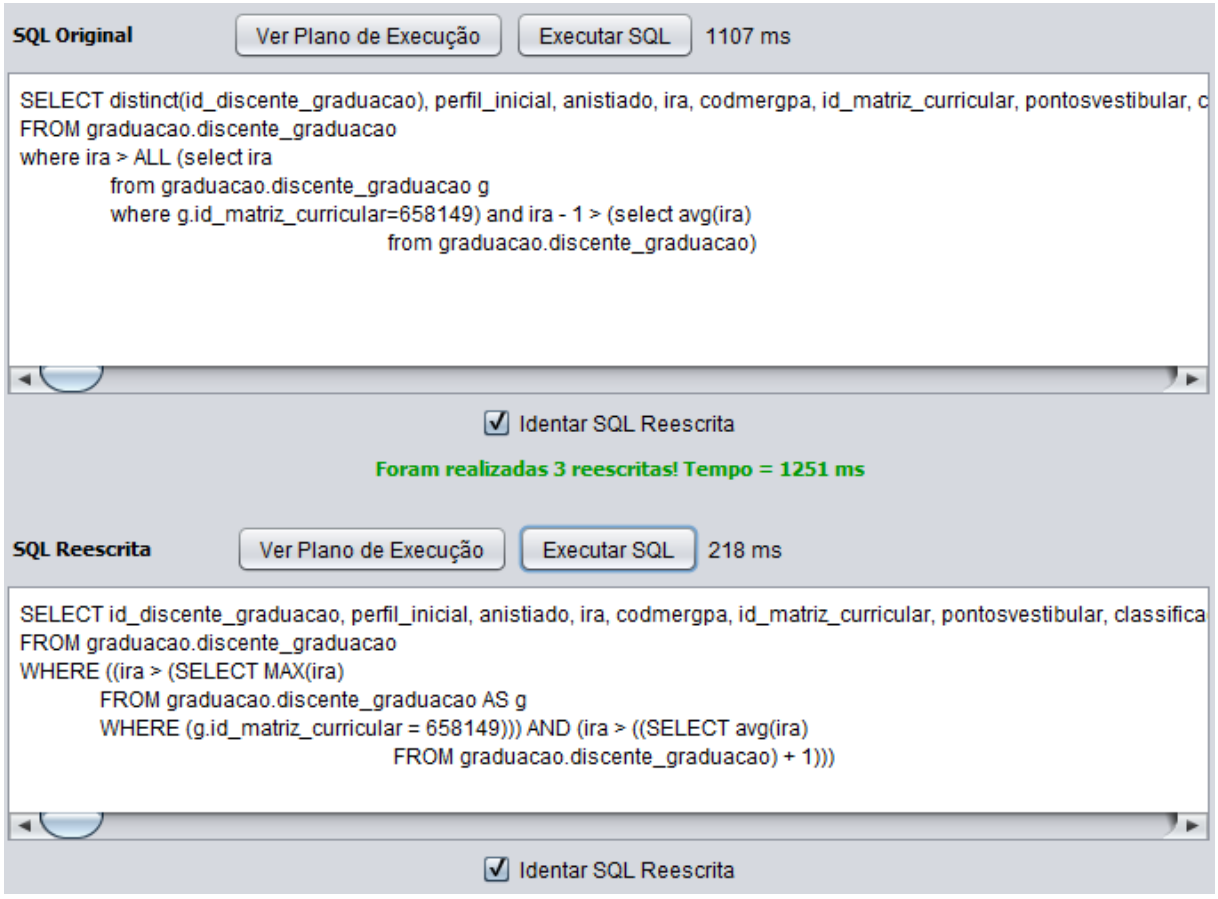

Figura 5.13: Tela para a seleção de uma instrução SQL original e visualização da instrução SQL reescrita.

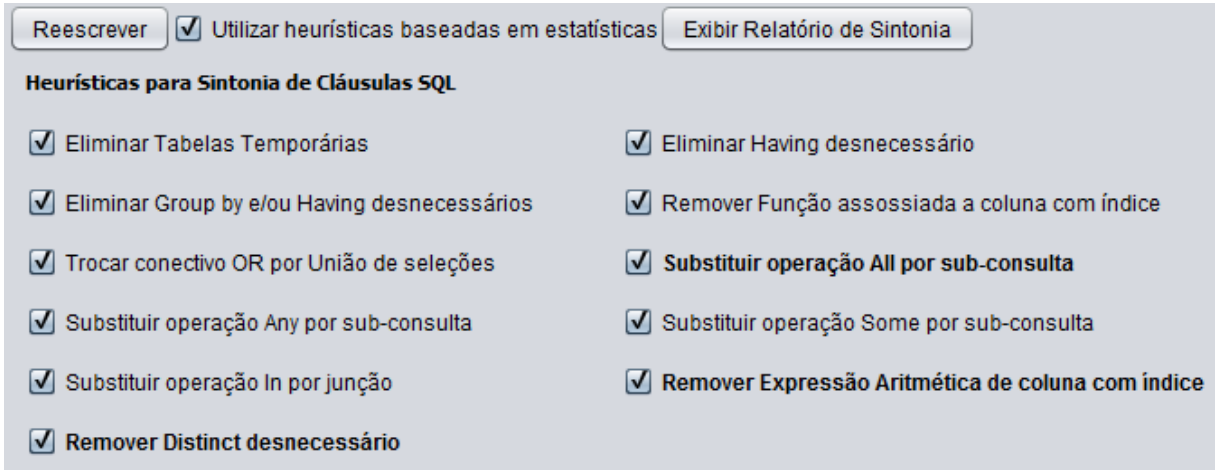

Figura 5.14: Tela de seleção e execução de heurísticas.

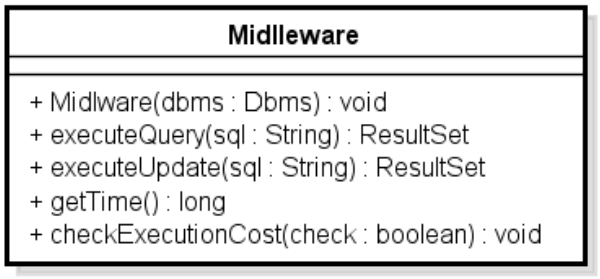

Figura 5.15: Classe Midlleware da Ferramenta Interactive Query Tuning.

em particular quanto para as instruções SQL em geral podem ser utilizadas pelo *Midlleware* que compõe a abordagem automática.

O *midlleware* desenvolvido utilizando a abordagem automática consiste uma classe Java denominada *Midlleware* (Figura 5.15). Esse classe pode ser utilizada pelos desenvolvedores de aplicação a fim de que uma determina instrução SQL seja reescrita antes de ser enviada ao SGBD. Assim, os desenvolvedores não mais utilizariam a API (*Application Programming Interface*) JDBC (pacote java.sql) diretamente, e sim por meio do *midlleware*.

A classe Midlleware possui um construtor que recebe como parâmetro um objeto da classe DBMS. Esse objeto irá conter as informações de acesso ao SGBD, como, por exemplo: endereço do servidor, usuário e senha. Os dois principais métodos da classe Midlleware são executeQuery e executeUpdade que executam uma instrução SQL fornecida como parâmetro. Porém, antes de executar uma instrução SQL, o *midlleware* verifica se a instrução possui oportunidades de sintonia e a reescreve (se necessário). Após esta etapa, a instrução SQL (reescrita ou original) é enviada ao SGBD. Após executar da instrução SQL, o método executeQuery retorna um objeto da classe ResultSet com a coleção de dados selecionados (se a instrução for uma seleção) e o método executeUpdade retorna a quantidade de elementos atualizadas ou excluídos (se a instrução for uma atualização ou exclusão). Opcionalmente, o método getTime pode ser utilizado para obter o tempo gasto na reescrita de uma instrução SQL em milissegundos. Além disso,o *middleware* pode checar se o custo de execução da instrução SQL original é menor que o da instrução reescrita, com a finalidade de garantir que o tempo de execução da instrução reescrita seja menor que o da original. Para isso, o método checkCostExecution deve receber o parâmetro *true*. Por padrão (*default*), esse método não verifica os custos de execução das instruções SQL.

6 RESULTADOS EXPERIMENTAIS

6.1 Ambiente de Execução

O ambiente de experimentação utilizado foi composto por uma estação Core i3-2100 3.10GHz, com 4GB de Ram e 500 GB de Hd, e com os sistemas operacionais Windows Seven e Linux Ubuntu. Os SGBDs utilizados nos testes foram os PostgreSQL 8.1 (no linux), Oracle 11g (no windows) e SQL Server 2008 (no windows). Optamos por utilizar o PostgreSQL instalado no sistema operacional Linux devido à sua baixa eficiência no sistema operacional Windows. Por exemplo, a execução das 21 consultas do *benchmark* TPC-H no PostgreSQL instalado no sistema operacional Windows durou aproximadamente 8,29 minutos, enquanto o mesmo teste quando executado no sistema operacional Linux durou aproximadamente 1,56 minutos.

Para auxiliar nas experimentações, utilizamos o dBest (database Benchmark test toolkit) (LEMOS; HOLANDA; MONTEIRO, 2011) o qual consiste em um *framework* multi-plataforma, *multi-benchmark* e multi-SGBD que fornece suporte para a realização de avaliações de desempenho em bancos de dados. O dBest fornece suporte para todas as etapas envolvidas no processo de avaliação de desempenho de bancos de dados, possibilita a rápida prototipagem de novos *benchmarks* e a fácil inclusão de novos SGBDs.

6.2 Cenários de Teste

A fim de mensurar a eficácia da abordagem proposta, utilizamos três cenários diferentes de testes. O primeiro cenário utiliza o *benchmark* TPC-H, o qual consiste em um *benchmark* voltado para aplicações de suporte a decisão. O TPC-H é formado por um conjunto de 22 consultas *ad-hoc* (além de dois comandos DDL utilizados para a criação e a remoção de uma visão) e possui uma estrutura padrão composta por oito tabelas. Destas, seis são tabelas dimensionais (Region, Nation, Supplier, Part, Customer e Partsupp) e duas de fatos (Orders e Lineitem). Para a realização dos testes foi utilizada uma base de dados de 1GB, onde a tabela Liniitem tem 6.000.000 de tuplas, por exemplo. No segundo cenário de testes utilizamos a base e dados do *benchmark* TPC-H e uma carga de trabalho sintética formada por 30 consultas SQL, onde cada consulta inclui uma ou mais possibilidades de sintonia. O terceiro cenário de teste utiliza a base de dados do SIG (Sistema Integrado de Gestão) da Universidade Federal do Ceará – UFC que é composto por dois módulos: SIGAA (Sistema Integrado de Gestão das Atividades Acadêmicas) e SIGRH (Sistema Integrado de Gestão de Recursos Humanos). A base de dados possui 36 es-

Tabela 6.1: Tempos de execução das consultas TPC-H 18 e 20 antes e após a reescrita de consultas.

	PostgreSQL		**SQL Server**	
	Original	**Reescrita**	**Original**	**Reescrita**
Consulta TPC-H 18	134.807ms	111.933ms	19s	14s
Consulta TPC-H 20	912ms	712ms	15s	11s

quemas e mais de 900 tabelas. Além disso, utilizamos uma carga de trabalho sintética formada por 30 consultas SQL, onde cada consulta apresenta pelo menos uma oportunidade de sintonia.

Para cada cenário, três testes foram executados: i) executou-se a carga de trabalho contendo as consultas originais (sem reescrita); ii) a carga de trabalho original (sem reescrita) foi submetida ao advisor (abordagem assistida) e, em seguida, a nova carga de trabalho (contendo as consultas ajustadas) foi executada; e iii) cada consulta da carga de trabalho original foi enviada para o *middleware* (abordagem automática), o qual procedeu a reescrita da consulta (quando necessário) e, em seguida, executou a consulta reescrita. Para cada teste executamos 1, 2, 4, 8, 16 e 32 iterações da carga de trabalho que foram submetidas de três formas distintas: sequencial, aleatória, aleatória fixa (onde define-se uma sequencia aleatória e depois mantem-se essa sequencia).

6.2.1 Cenário 1: Benchmark TPC-H

Das 23 consultas que compõem o *benchmark* TPC-H, duas (consultas 18 e 20) foram reescritas (sintonizadas) pelo agente AST (Figura 5.1), por meio da heurística H8 (Tabela 4.1), para os SGBDs SQL Server e PostgreSQL. A Tabela 6.1 mostra os tempos de execução das consultas TPC-H 18 e 20 em seus formatos originais e após serem reescritas. Observando as Figuras 6.1, 6.2, 6.3, 6.4, 6.5 e 6.6 podemos concluir que a abordagem assistida apresentou uma pequena diminuição no tempo de execução da carga de trabalho. O que é explicado pelo fato de somente duas consultas do TPC-H terem apresentado oportunidades de sintonia. Porém, essas duas consultas não foram reescritas pelo agente AST para o Oracle, uma vez que este já implementa a heurística que possibilitou reescrever as duas consultas do TPC-H. Assim, observamos nas Figuras 6.7, 6.7 e 6.9 que a abordagem automática apresentou uma leve piora em relação ao *baseline*. O que é explicado pelo fato da abordagem automática ter tido o *overhead* de tentar reescrever cada uma das consultas recebidas e nenhuma delas possuir oportunidade de sintonia no Oracle.

A Figura 6.2 ilustra o resultado do teste aleatório com o *benchmark* TPC-H no PostgreSQL. Neste experimento, a execução das 32 iterações da carga de trabalho original (sem reescrita) demorou 304 segundos. Já a execução das 32 iterações da carga de trabalho gerada após a aplicação da abordagem assistida teve duração de 283 segundos. Neste caso, a abordagem assistida proporcionou um ganho de 21 segundos.

A Figura 6.6 ilustra o resultado do teste aleatório fixo com o *benchmark* TPC-H no SQL Server. Neste experimento, a execução das 32 iterações da carga de trabalho original (sem reescrita)

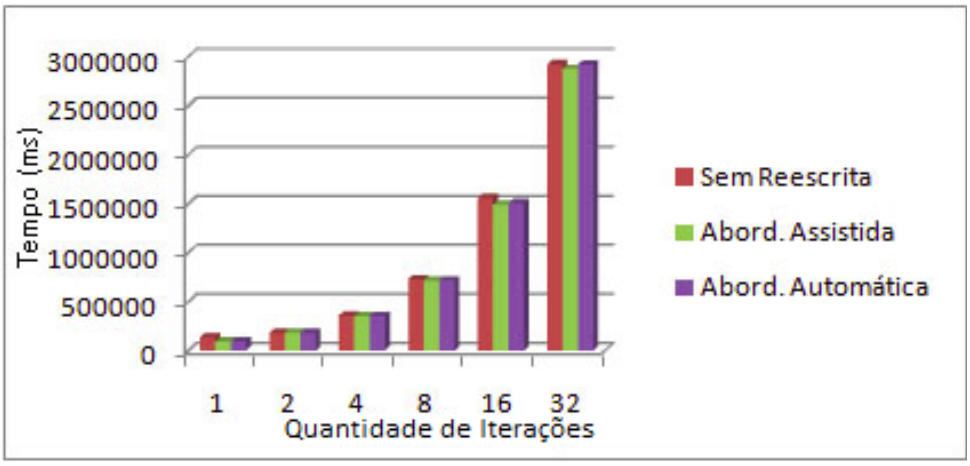

Figura 6.1: Teste sequencial com o *benchmark* TPC-H no PostgreSQL.

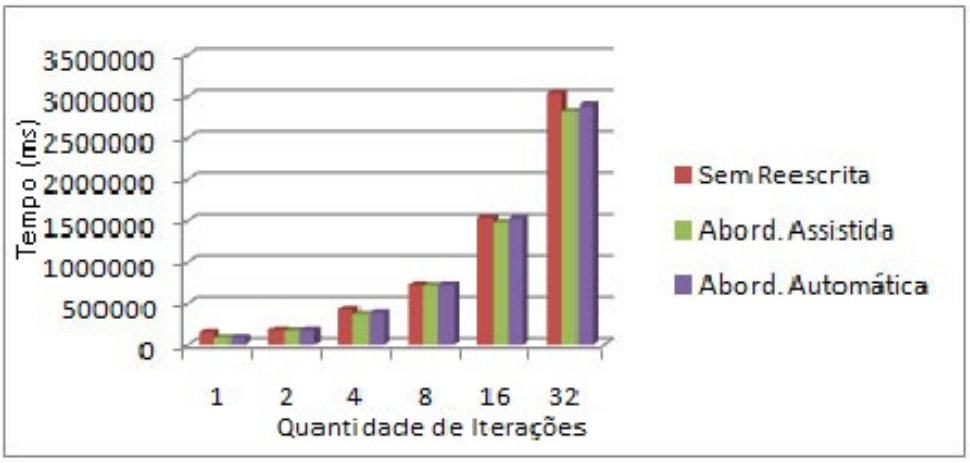

Figura 6.2: Teste aleatório com o *benchmark* TPC-H no PostgreSQL.

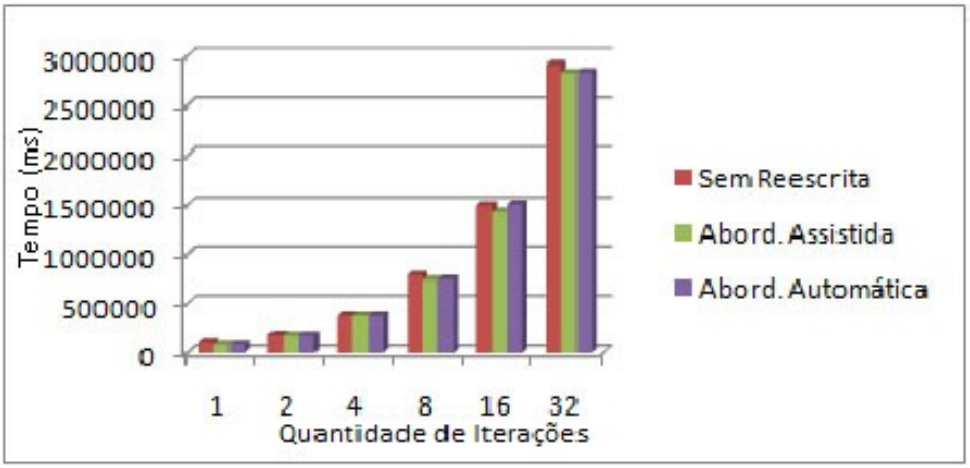

Figura 6.3: Teste aleatório fixo com o *benchmark* TPC-H no PostgreSQL

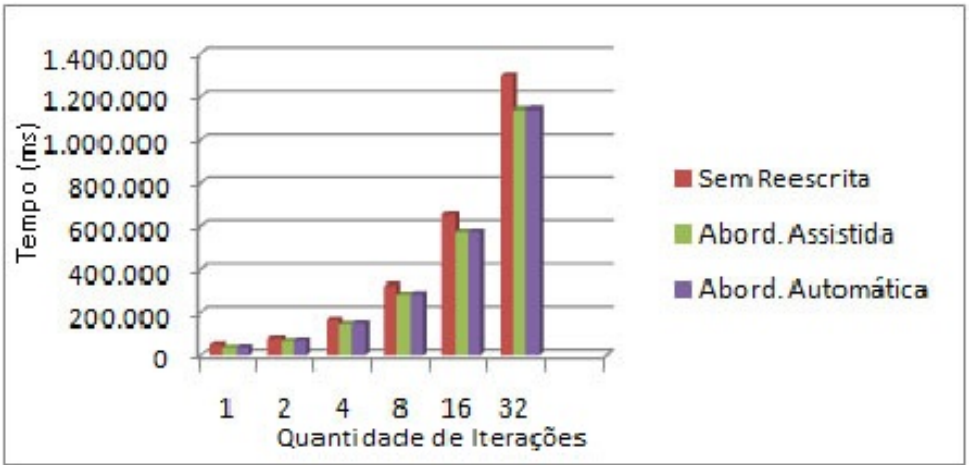

Figura 6.4: Teste sequencial com o *benchmark* TPC-H no SQL Server.

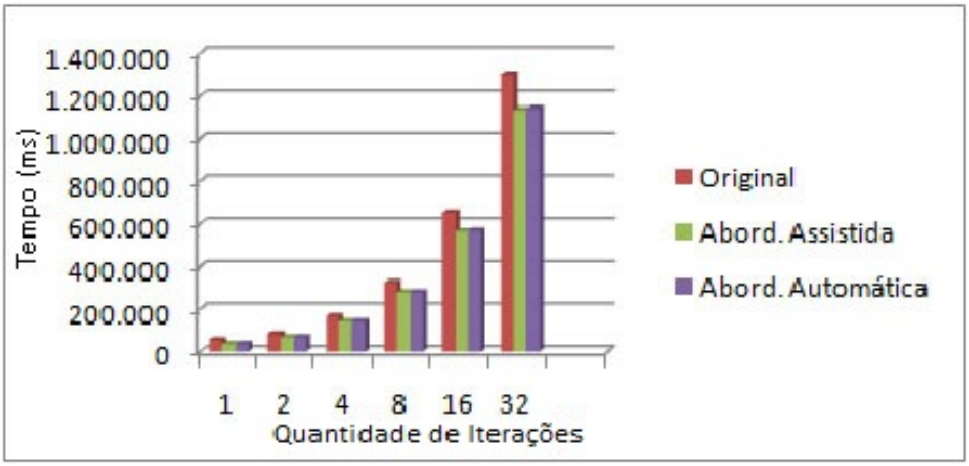

Figura 6.5: Teste aleatório com o *benchmark* TPC-H no SQL Server.

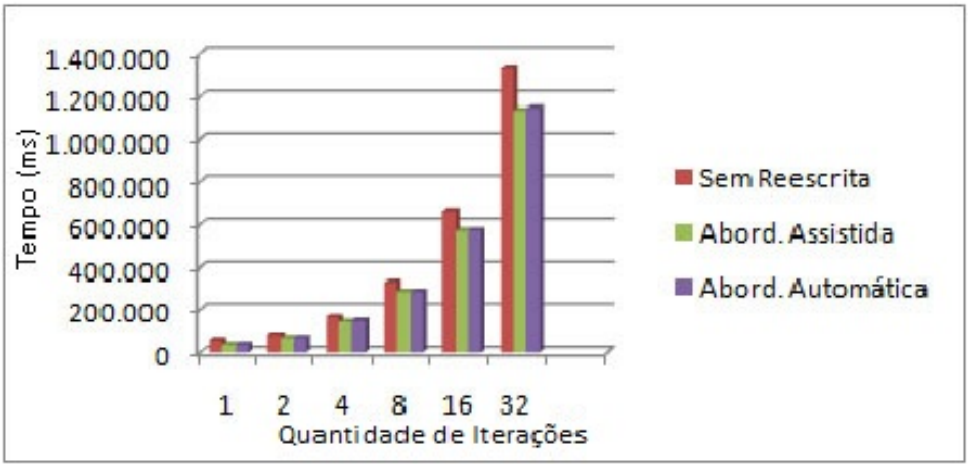

Figura 6.6: Teste aleatório fixo com o *benchmark* TPC-H no SQL Server.

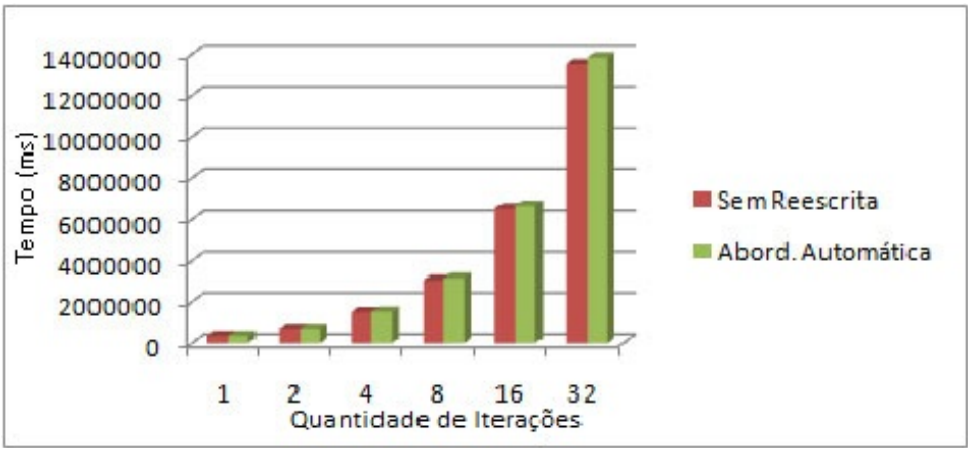

Figura 6.7: Teste sequencial com o *benchmark* TPC-H no Oracle.

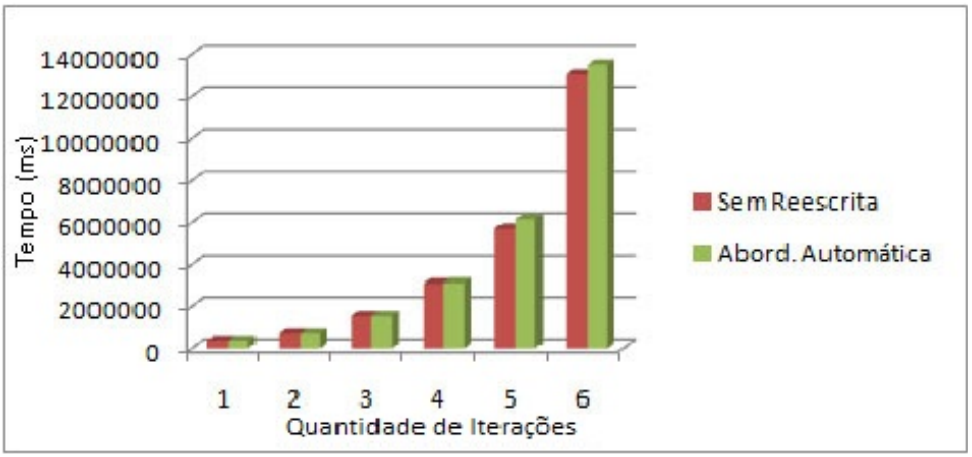

Figura 6.8: Teste aleatório com o *benchmark* TPC-H no Oracle.

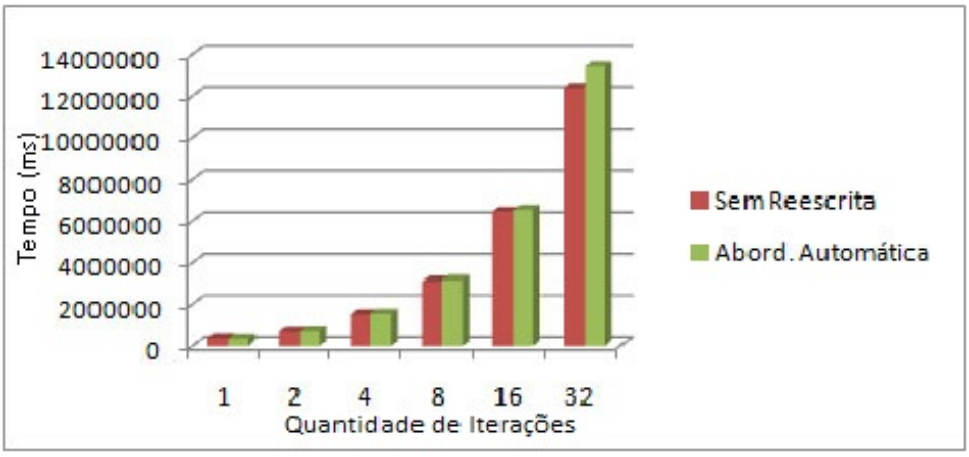

Figura 6.9: Teste aleatório fixo com o *benchmark* TPC-H no Oracle.

demorou 133,64 segundos. Já a execução das 32 iterações da carga de trabalho gerada após a aplicação da abordagem assistida teve duração de 113,63 segundos. Neste caso, a abordagem assistida proporcionou um ganho de 20 segundos.

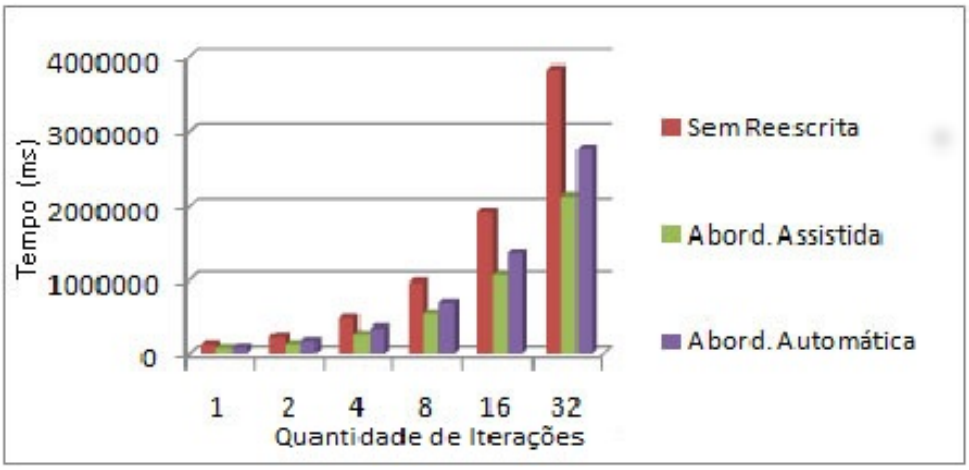

Figura 6.10: Teste sequencial com 30 consultas sintéticas na base TPC-H no PostgreSQL.

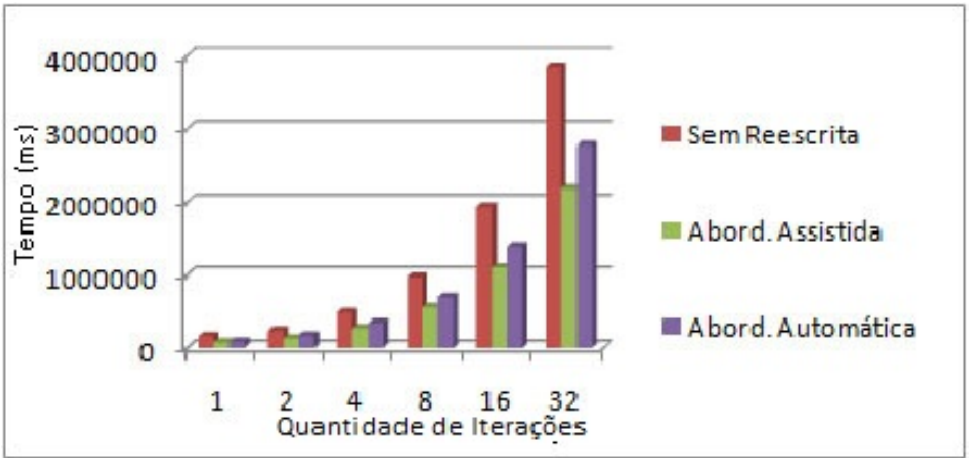

Figura 6.11: Teste aleatório com 30 consultas sintéticas na base TPC-H no PostgreSQL.

6.2.2 Cenário 2: Base de Dados do TPC-H + Carga de Trabalho Sintética

As Figuras 6.10, 6.11, 6.12, 6.13, 6.14, 6.15, 6.16, 6.17 e 6.18 mostram que a abordagem assistida proporcionou uma grande redução no tempo de execução das cargas de trabalho. Já a abordagem automática proporcionou benefícios menores uma vez que esta envolve o *overhead* de ajustar as instruções SQL recebidas em tempo de execução. Vale destacar que nos testes realizados no Oracle (Figuras 6.16, 6.17 e 6.18) a abordagem automática apresentou apenas uma pequena diminuição no tempo de execução da carga de trabalho. Das onze heurísticas disponíveis na abordagem, cinco já são implementadas pelo Oracle (H4, H5, H6, H7 e H8). E das seis heurísticas que restaram para ajustar a carga de trabalho, três delas (H9, H10 e H11) fazem uso de informações estatísticas e, consequentemente, necessitam de acessos adicionais ao SGBD aumentado significativamente o *overhead* ao ajustar as instruções SQL.

A Figura 6.12 ilustra o resultado do teste aleatório fixo com o 30 consultas sintéticas na base TPC-H no PostgreSQL. Neste experimento, a execução das 32 iterações da carga de trabalho original (sem reescrita) demorou 391,68 segundos. Já a execução das 32 iterações da carga de trabalho gerada após a aplicação da abordagem assistida teve duração de 222,46 segundos.

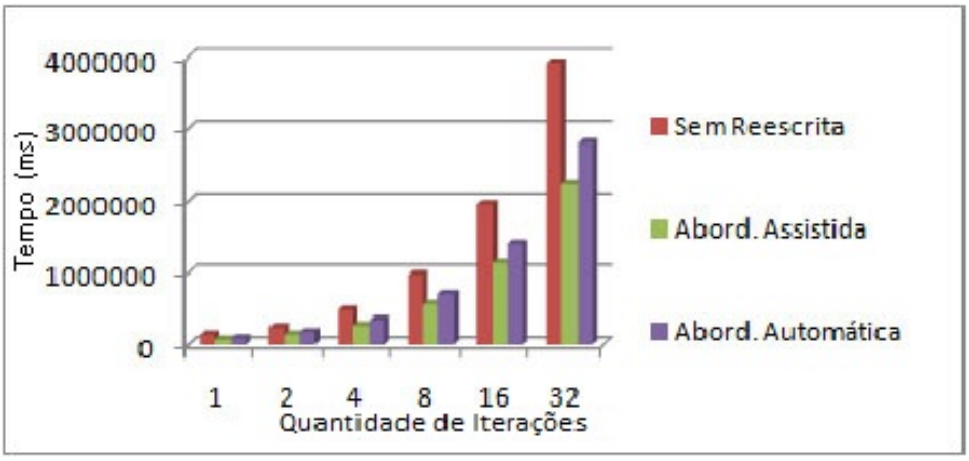

Figura 6.12: Testes aleatório fixo com 30 consultas sintéticas na base TPC-H no PostgreSQL.

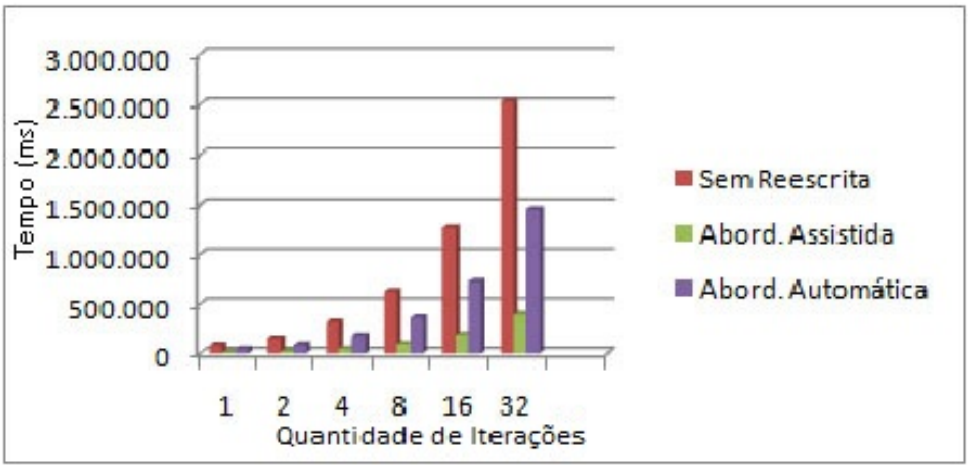

Figura 6.13: Teste sequencial com 30 consultas sintéticas na base TPC-H no SQL Server.

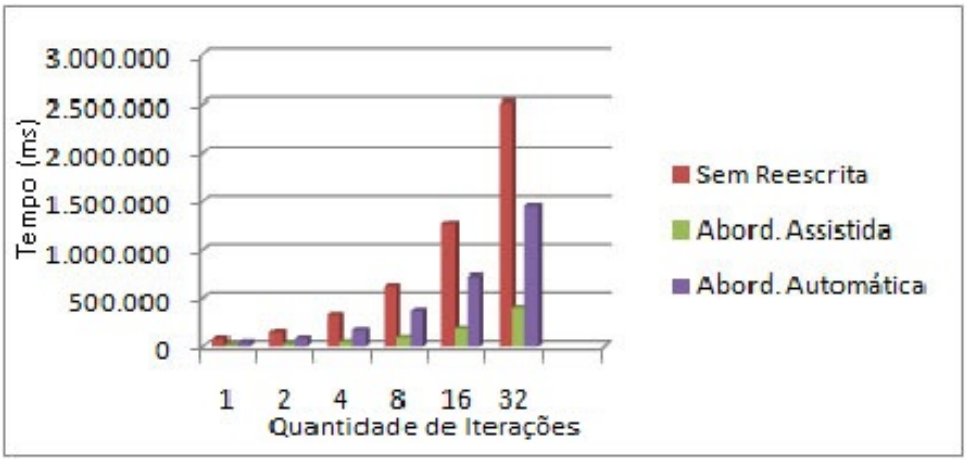

Figura 6.14: Testes aleatórios das 30 consultas sintéticas na base TPC-H do SQL Server.

Neste caso, a abordagem assistida proporcionou um ganho de 172 segundos.

A Figura 6.14 ilustra o resultado do teste aleatório com o 30 consultas sintéticas na base TPC-H no SQLServer. Neste experimento, a execução das 32 iterações da carga de trabalho original

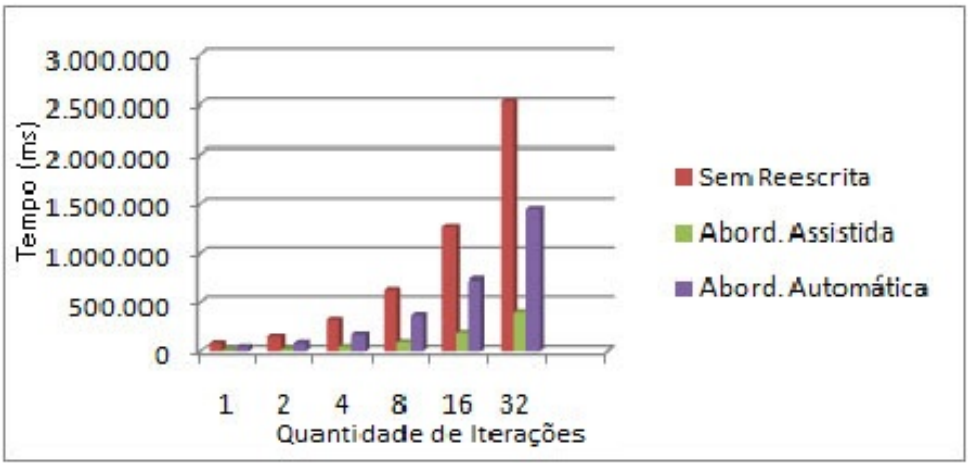

Figura 6.15: Teste aleatório fixo com 30 consultas sintéticas na base TPC-H no SQL Server.

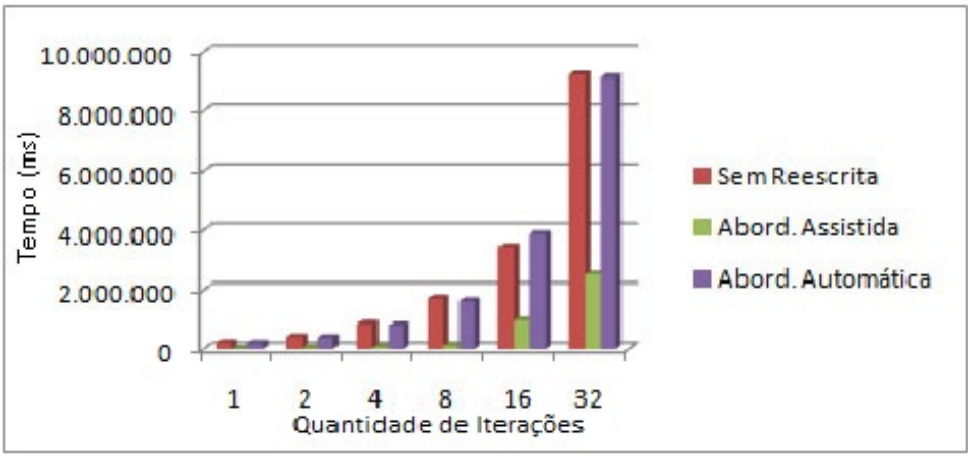

Figura 6.16: Teste sequencial com 30 consultas sintéticas na base TPC-H no Oracle.

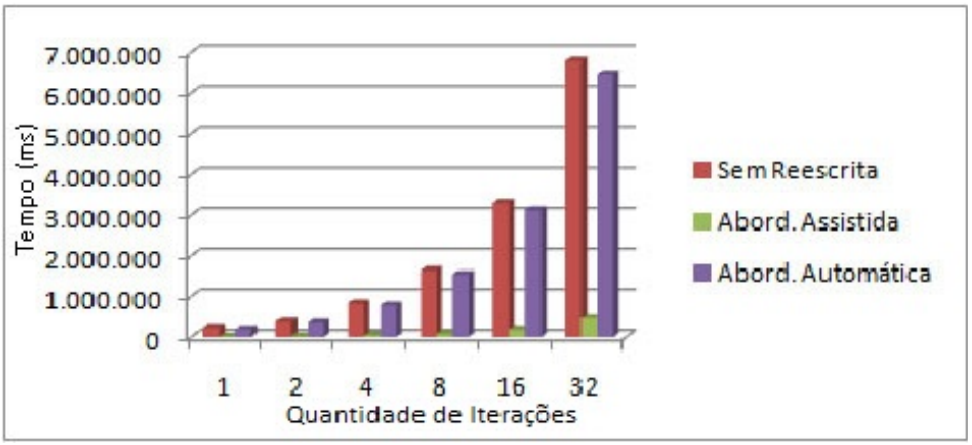

Figura 6.17: Teste aleatório com 30 consultas sintéticas na base TPC-H no Oracle.

(sem reescrita) demorou 385,68 segundos. Já a execução das 32 iterações da carga de trabalho gerada após a aplicação da abordagem assistida teve duração de 218,36 segundos. Neste caso, a abordagem assistida proporcionou um ganho de 167 segundos.

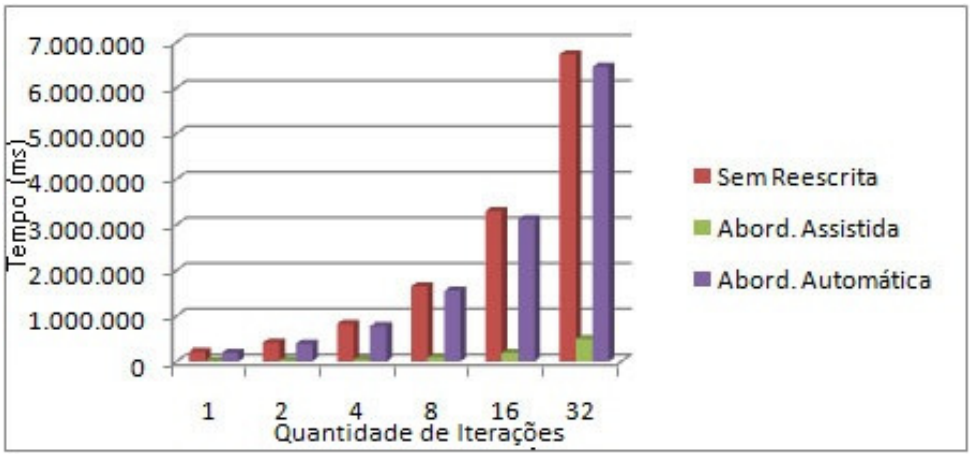

Figura 6.18: Teste aleatório fixo com 30 consultas sintéticas na base TPC-H no Oracle.

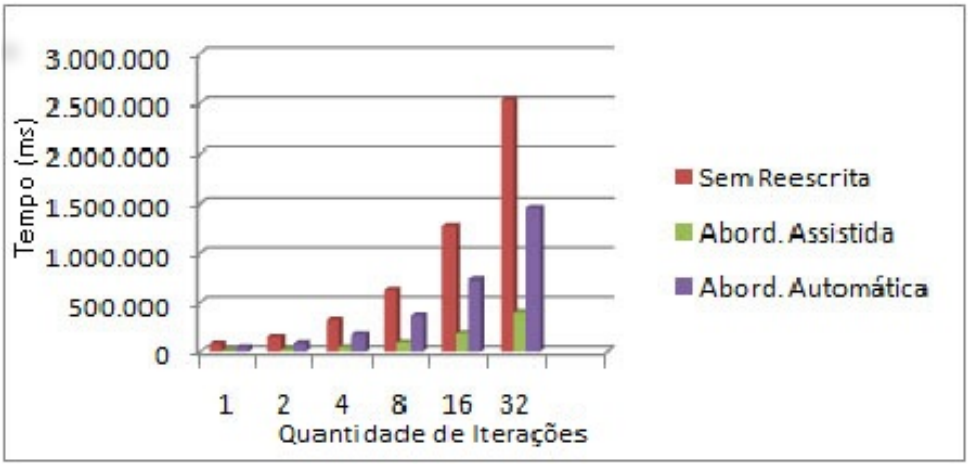

Figura 6.19: Teste sequencial na base SIG no PostgreSQL.

A Figura 6.18 ilustra o resultado do teste aleatório fixo com o 30 consultas sintéticas na base TPC-H no Oracle. Neste experimento, a execução das 32 iterações da carga de trabalho original (sem reescrita) demorou 673,28 segundos. Já a execução das 32 iterações da carga de trabalho gerada após a aplicação da abordagem assistida teve duração de 49,43 segundos. Neste caso, a abordagem assistida proporcionou um ganho de 623 segundos. Já o mesmo teste com a abordagem automática demorou 645,95. Esse resultado decorre do fato que a maioria das heurísticas que o Oracle ainda implementa (H1, H2, H3, H9, H10 e H11) utilizam operações estatísticas, o que implica na execução de consultas adicionais para recuperar essas informações do SGBD.

6.2.3 Cenário 3: Base do SIG + Carga de Trabalho Sintética

Este cenário foi avaliado somente no PostgreSQL uma vez que a base de dados do SIG está disponível somente neste SGBD. Novamente, as abordagens propostas proporcionaram uma elevada redução no tempo de execução das cargas de trabalho (Figuras 6.19, 6.20 e 6.21).

A Figura 6.20 ilustra o resultado do teste aleatório com o 30 consultas sintéticas na base SIG

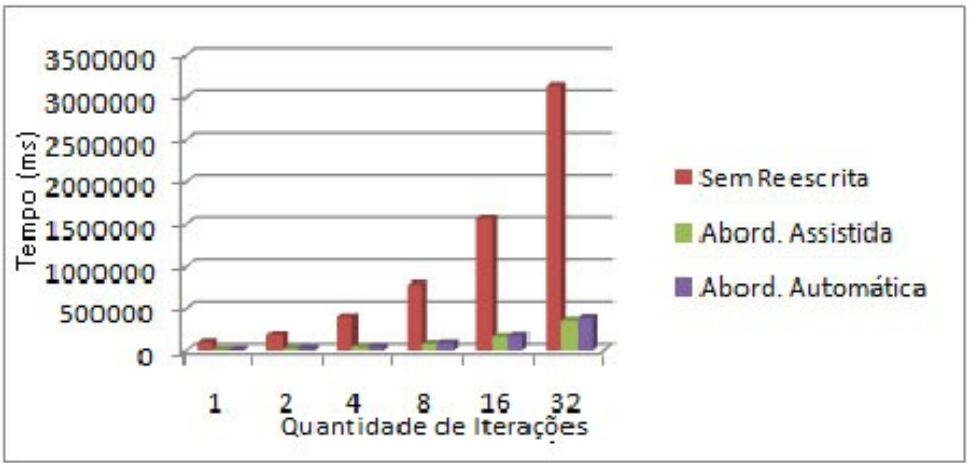

Figura 6.20: Teste aleatório na base SIG no PostgreSQL.

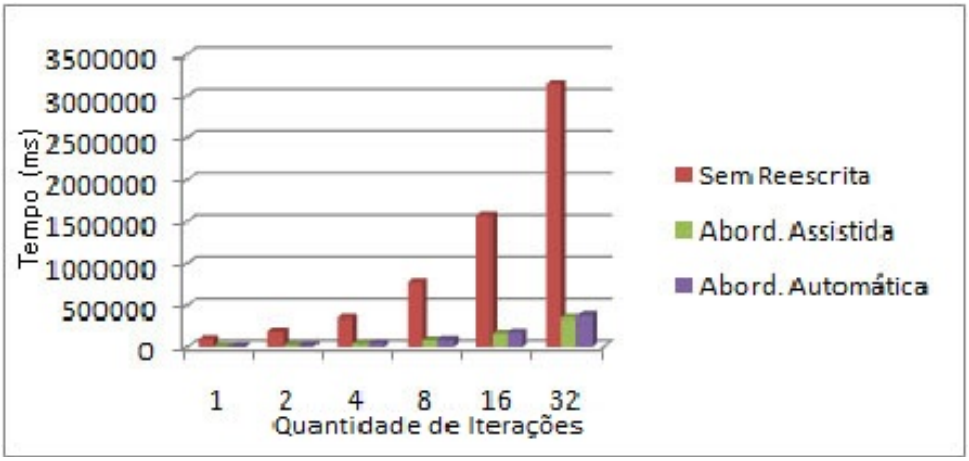

Figura 6.21: Teste aleatório fixo na base SIG no PostgreSQL.

no PostgreSQL. Neste experimento, a execução das 32 iterações da carga de trabalho original (sem reescrita) demorou 314,32 segundos. Já a execução das 32 iterações da carga de trabalho gerada após a aplicação da abordagem assistida teve duração de 34,72 segundos. Neste caso, a abordagem assistida proporcionou um ganho de 280 segundos.

6.3 Comparação entre a ferramenta *Interactive Query Tuning* e a ferramenta *Quest SQL Optimizer for Oracle*

A *Quest SQL Optimizer for Oracle* é uma ferramenta capaz de identificar oportunidades de sintonia em instruções SQL e ajustar esses comandos utilizando tanto a reescrita de instruções SQL quanto a aplicação de *hints*. Dada uma determinada instrução SQL Q, essa ferramenta gera várias instruções equivalentes a Q, reescrevendo-a e/ou aplicando *hints*, e estima o custo de execução de cada uma dessas instruções. Assim, o DBA pode comparar o desempenho da instrução SQL original com o desempenho das instruções geradas pela ferramenta. A ferramenta também possibilita executar as todas instruções SQL geradas no processo de sintonia

de *Q* para obter o tempo de execução real de cada uma delas. Contudo, essa opção não foi avaliada, uma vez que somente para a consulta 5, por exemplo, 228 comandos SQL diferentes precisariam ser executados.

Com a finalidade de avaliar a ferramenta *Quest SQL Optimizer for Oracle* e compará-la com a ferramenta *Interactive Query Tuning*, proposta neste trabalho, realizamos um experimento utilizando o *benchmark* TPC-H. Das 22 consultas do *benchmark* TPC-H utilizamos neste teste 21 consultas. Uma das consultas do TPC-H foi desprezada por constituir-se de um comando SQL baseado na utilização de uma visão (consulta 23 do TPC-H), a qual é criada e removida por dois comandos DDL pertencentes ao TPC-H, comandos 15 e 24, respectivamente. Essa estratégia foi utilizada exclusivamente para facilitar a execução dos experimentos. Assim, Os experimentos foram realizados no mesmo ambiente descrito na Seção 6.1. Os resultados dos testes estão sumarizados na Tabela 6.2. Em cada linha da Tabela 6.2 pode-se encontrar: o identificador da consulta, o número de instruções SQL geradas pela ferramento durante o processo de sintonia, o tempo gasto no processo de sintonia, o número de instruções SQL geradas cuja estimativa de custo é menor que a estimativa de custo do comando original, o número de instruções SQL geradas cuja estimativa do tempo de execução é menor que a estimativa do tempo de execução do comando original, o tempo de execução da instrução SQL original e o tempo de execução da melhor instrução SQL gerada pela ferramenta (ou seja, da instrução SQL com menor estimativa para o tempo de execução).

Vale destacar que a ferramenta *Quest SQL Optimizer for Oracle* conseguiu ajustar corretamente as consultas 2, 3, 11, 12 e 20 (5 consultas no total) por meio da aplicação de *hints*. Por outro lado, a ferramenta *Interactive Query Tuning* não reescreveu nenhuma das 22 consultas do *benchmark* TPC-H no Oracle (somente no PostgreSQL e SQLServer).

O relatório gerado pela ferramenta *Quest SQL Optimizer for Oracle* indicou que das 22 consultas analisadas apenas 6 consultas não poderiam ser ajustadas, ou seja, a ferramenta não conseguiu gerar uma instrução SQL equivalente (reescrita) que proporcionasse um tempo de execução estimado melhor que o tempo de execução estimado da instrução original. Foram essas as consultas 1, 6, 10, 13, 17 e 19. Logo, o relatório indicou que 16 consultas poderiam ser otimizadas com sucesso.

O próximo passo do experimento consistiu em executar (com *cache* vazio) para cada uma dessas 16 consultas a instrução SQL original e a melhor instrução SQL equivalente gerada pelo *Quest SQL Optimizer for Oracle*. O resultado desse experimento mostrou que das 16 consultas ajustadas apenas 5 tiveram seu tempo de execução reduzido (foram elas as consultas 2, 3, 11, 12 e 20). Já as consultas 4, 5, 7, 8, 9, 14, 16, 18, 21 e 22 tiveram, na prática, uma piora nos seus tempos de execução. Logo, a ferramenta errou em 11 consultas e acertou em apenas 5 comandos SQL, de um total de 16 instruções SQL. Vale observar que em alguns casos, como nas consultas 4 e 7, o tempo de resposta da instrução SQL gerada pela ferramenta da **Quest** foi muito maior que o tempo de resposta da consulta original, piorando bastante o desempenho dessas consultas. No apêndice mostramos com mais detalhes os resultados obtidos nesse experimento.

Adicionalmente, a ferramenta *Quest SQL Optimizer for Oracle* possui a limitação de poder ser

Tabela 6.2: Resultados obtidos nos testes realizados com a ferramenta *Quest SQL Optimizer for Oracle* utilizando o *benchmark* TPC-H.

Consulta	Nº de instruções SQL geradas na sintonia	Tempo gasto na sintonia	Nº de instruções com estimativa de custo menor que o da consulta original	Nº de consultas com tempo de execução estimado menor que o tempo de execução estimado da consulta original	Tempo de execução da consulta original	Tempo de execução da melhor instrução SQL gerada
1	0	1s	-	-	8210ms	-
2	115	116s	3	3	310ms	128ms
3	53	58s	3	1	6393ms	4900ms
4	14	15s	1	0	4204ms	+ de 1 min
5	228	229s	5	0	4658ms	4666ms
6	1	1s	0	0	2313ms	-
7	110	111s	1	0	4658ms	7808ms
8	140	141s	1	0	4567ms	4770ms
9	191	192s	5	0	6181ms	6749 ms
10	118	120s	0	0	3160ms	-
11	139	140s	8	4	600ms	259ms
12	20	21s	1	1	1995ms	1993
13	2	3	0	0	1995ms	-
14	5	6s	1	0	512ms	538ms
16	35	36s	2	0	278.244ms	280.190 ms
17	29	30s	0	0	1.635ms	-
18	160	161s	1	0	17.384ms	22.306 ms
19	86	87s	0	0	3.228ms	-
20	92	93s	8	3	3.261ms	3.212ms
21	120	121s	5	0	8.660ms	8.787ms
22	1	1s	1	0	407ms	418ms

utilizada apenas com o SGBD Oracle. Além disso, a ferramenta pode gerar algumas dezenas ou centenas de instruções para ajustar uma determinada instrução SQL, tornando o seu tempo de sintonia bastante demorado. Isso inviabiliza a utilização da ferramenta em cenários onde a instrução SQL precisa ser ajustada automaticamente, ou seja, em tempo de execução.

A ferramenta *Interactive Query Tuning* pode ser utilizada com qualquer SGBD (desde que hajam *drivers* implementados para o SGBD a ser utilizando); e pode atuar como um *midleware* entre as aplicações e o SGBD, ajustando uma instrução SQL (se necessário) em tempo de execução.

6.4 Overhead de execução das heurísticas para sintonia de cláusulas SQL.

As heurísticas pesquisadas e implementadas nesse trabalho (ver sessão) podem demandar algum tempo para serem executadas e, adicionalmente, influenciar no tempo de execução das cláusulas SQL na abordagem automática. Entretanto, raramente todas as heurísticas são realmente utilizadas para reescrever uma cláusula SQL. Dessa forma, se apenas as heurísticas que realmente podem reescrever uma determinada cláusula SQL forem utilizadas, o tempo de reescrita da cláusula pode diminuir consideravelmente. Para constatar isso, fizemos um experimento utilizado as consultas do teste realizado na sessão 6.2.2 (ver também apêndice D). Para cada uma dessas consultas, foram coletados o tempo de execução de todas heurísticas para sintonia de cláusulas SQL e o tempo de execução de apenas as heurísticas que realmente reescreviam a consulta. A Tabela 6.3 exibe os resultados do experimento para os SGBDs PostgreSQL, SQL Server e Oracle.

Tabela 6.3: Experimento para verificar o *overhead* de execução de todas as heurísticas em relação a execução de apenas as heurísticas que realmente reescrevem uma deteminada consulta.

	Todas as heurísticas			**Heurísticas realmente utilizadas**		
	PostgreSQL	SQL Server	Oracle	PostgreSQL	SQL Server	Oracle
01	44ms	18ms	28ms	0ms	0ms	-
02	38ms	18ms	30ms	0ms	0ms	-
03	40ms	18ms	24ms	0ms	0ms	-
04	39ms	19ms	27ms	0ms	0ms	-
05	44ms	17ms	24ms	0ms	0ms	-
06	38ms	21ms	26ms	0ms	0ms	-
07	27ms	13ms	17ms	27ms	13ms	17ms
08	18ms	10ms	18ms	18ms	10ms	18ms
09	100ms	50ms	80ms	64ms	30ms	36ms
10	58ms	23ms	42ms	40ms	19ms	29ms
11	60ms	27ms	42ms	38ms	19ms	25ms
12	0ms	0ms	0ms	0ms	0ms	0ms
13	25ms	9ms	16ms	22ms	9ms	15ms
14	43ms	17ms	26ms	0ms	0ms	0ms
15	40ms	17ms	30ms	23ms	9ms	15ms
16	76ms	37ms	45ms	23ms	10ms	15ms
17	59ms	28ms	36ms	23ms	9ms	15ms
18	22ms	9ms	13ms	19ms	9ms	11ms
19	23ms	9ms	14ms	23ms	9ms	14ms
20	20ms	9ms	15ms	20ms	9ms	15ms
21	40ms	18ms	24ms	39ms	18ms	24ms
22	38ms	17ms	23ms	38ms	17ms	23ms
23	38ms	17ms	25ms	38ms	17ms	25ms
24	38ms	16ms	33ms	0ms	0ms	0ms
25	42ms	16ms	39ms	0ms	0ms	0ms
26	40ms	16ms	45ms	0ms	0ms	0ms
27	55ms	25ms	38ms	20ms	9ms	14ms
28	77ms	36ms	43ms	72ms	33ms	42ms
29	59ms	25ms	38ms	24ms	9ms	16ms
30	55ms	25ms	34ms	20ms	8ms	16ms

7 CONCLUSÕES E TRABALHOS FUTUROS

A sintaxe de um comando SQL influencia a escolha do plano de execução. Consequentemente, comandos SQL equivalentes, que possuem sintaxe diferente mas a mesma semântica e o mesmo resultado, podem apresentar tempos de resposta diferentes. Por este motivo, uma das principais atividades realizadas pelos DBAs com a finalidade de melhorar o desempenho de um determinado comando SQL consiste no ajuste (ou sintonia) desse comando. Para isso, em geral, duas estratégias são frequentemente utilizadas: (a) reescrever o comando SQL; e (b) aplicar *Query Hinting* (SHASHA; BONNET, 2003).

Dado um determinado comando SQL Q_1. A técnica de reescrita consiste em escrever um novo comando SQL Q_2, equivalente ao comando SQL original Q_1, com o objetivo de que o tempo de resposta apresentado por Q_2 seja menor que o tempo de resposta de Q_1 ($TR_{Q_2} < TR_{Q_1}$), o que pode ser obtido caso o plano de execução gerado pelo otimizador de consultas do SGBD para a consulta Q_2 seja mais eficiente que o plano de execução gerado para Q_1.

A aplicação de *hints* consiste em um mecanismo comumente encontrado nos sistemas de bancos de dados comerciais. Essencialmente, um *hint* instrui o otimizador a restringir seu espaço de busca para certo subconjunto de planos de execução (por exemplo, impor a escolha de planos que usem um determinado tipo índice ou determinando a ordem e/ou método de junção). O uso de um *hint* é pode ser realizado apenas concatenado-o ao comando SQL.

Embora a sintonia de comandos SQL possa proporcionar ganhos de desempenho expressivos para as aplicações que utilizam bancos de dados relacionais, sua aplicação é complexa e requer bastante conhecimento em diversas áreas (HERODOTOU; BABU, 2009). Desta forma, fornecer ferramentas que auxiliem o DBA na difícil e repetitiva tarefa de ajustar comandos SQL torna-se de fundamental importância. Recentemente, algumas ferramentas têm sido disponibilizadas com o objetivo de auxiliar o DBA por meio de recomendações de sintonia de comandos SQL, tais como: Optim Development Studio da IBM (STUDIO, 2010), DB Optimizer XE da Embarcadero (OPTIMIZER, 2010), SQL Optimizer for Oracle da Quest (QUEST, 2010) e a Sql Tuning Advisor da Oracle (DAGEVILLE; DIAS, 2006). Contudo, em geral, essas ferramentas adotam uma abordagem *offline* (executam suas tarefas apenas quando são explicitamente acionadas pelo DBA), são específicas para um determinado SGBD e transferem para o DBA, dentre outras tarefas, a decisão de executar ou não as recomendações sugeridas.

Neste trabalho, propomos duas abordagens distintas para suportar a reescrita de comandos SQL em bancos de dados relacionais: uma abordagem assistida e outra automática. A abordagem assistida procura envolver o DBA (especialista em sintonia de instruções SQL) na solução do problema, com o objetivo de tirar proveito de sua experiência e conhecimento para proporcionar

bons resultados. A abordagem automática, por sua vez, busca realizar a sintonia de comandos SQL de forma completamente independente da interação com o DBA. Assim, a abordagem automática é direcionada para cenários onde um DBA não está disponível ou sua interferência não seria viável, como por exemplo, aplicações embarcadas ou disponibilizadas em plataformas de computação em nuvem.

As duas abordagens propostas utilizam um conjunto de heurísticas para realizar a reescrita dos comandos SQL. As heurísticas são constituídas de regras que visam identificar oportunidades de sintonia nos comandos SQL. Com o objetivo de avaliar a eficiência das abordagens propostas uma avaliação experimental foi realizada. Os experimentos foram conduzidos em três diferentes cenários: i) com o *benchmark* TPC-H, ii) com a base de dados do TPC-H e uma carga de trabalho sintética e iii) com a base de dados do sistema SIG e uma carga de trabalho sintética. Para cada cenário, três SGBDs foram avaliados: PostgreSQL, Oracle e SQL Server. Os resultados dos testes realizados mostram que tanto a abordagem assistida quanto a automática proporcionaram ganhos de desempenho, reduzindo o tempo de resposta das cargas de trabalho avaliadas. Assim, apesar da aplicação das heurísticas de reescrita de instruções SQL não assegurar a obtenção de uma solução ótima, ou seja, não garantir que a instrução SQL produzida é aquela que proporciona o menor tempo de resposta, os resultados comprovam o benefício proporcionada pela utilização dessas heurísticas.

7.1 Principais Contribuições

As principais contribuições desta dissertação foram:

1. A identificação de um conjunto de heurísticas que podem ser utilizadas para a reescrita de instruções SQL, de forma computacional;
2. Uma abordagem para a sintonia assistida de comandos SQL;
3. Uma abordagem para a sintonia automática de instruções SQL;
4. Uma arquitetura baseadas em agentes de *Software* para a implementação da abordagem assistida;
5. Uma arquitetura baseadas em agentes de *Software* para a implementação da abordagem automática;
6. Uma implementação da abordagem assistida e da abordagem automática. As arquiteturas e as abordagens propostas foram implementadas inteiramente em linguagem Java. O protótipo implementado fornece suporte para os SGBDs PostgreSQL 8.4, Oracle 11g e SQL Server 2008. Adicionalmente, outros SGBDs podem ser facilmente adicionados ao protótipo.

7.2 Oportunidades para Trabalhos Futuros

A seguir descrevemos algumas oportunidades para trabalhos futuros:

- Estender o protótipo implementado nessa dissertação a outros SGBDs, como MySQL, DB2, entre outros.
- Pesquisar e implementar novas heurísticas para a reescrita de instruções SQL.
- Pesquisar e implementar heurísticas que utilizem a técnica de aplicação de *hints* para ajustar instruções SQL.
- Investigar estratégias que permitam ao DBA (ou ao administrador do sistema) adicionar novas heurísticas para sintonia de instruções SQL sem a necessidade de se alterar o código fonte da ferramenta.
- Desenvolver no protótipo implementado a utilização de *cashs* para diminuir ainda mais o tempo de execução da cláusula SQL reescritas na abordagem automática. Um das *cashs* consiste em, após aplicar as heurísticas em uma determinada cláusula SQL, guardar quais heurísticas são realmente aplicáveis aquela cláusula, e aplicar na cláusula apenas essas heurísticas em execuções posteriores. A outra *cashs* consiste em, após reescrever uma heurística de uma determinada cláusula SQL, gravar sua reescrita para que execuções posteriores posteriores da cláusula não seja mais necessário executar as heurísticas novamente.

REFERÊNCIAS BIBLIOGRÁFICAS

ALAGIANNIS, I. et al. An automated, yet interactive and portable db desig. In: *SIGMOD '10 Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. New York, USA: [s.n.], 2010.

ALAGIANNIS, I. et al. An automated, yet interactive and portable db designer. In: *Proceedings of the 2010 international conference on Management of data*. New York, NY, USA: ACM, 2010. (SIGMOD '10), p. 1183–1186. ISBN 978-1-4503-0032-2. Disponível em: <http://doi.acm.org/10.1145/1807167.1807314>.

BRUNO, N. *Automated Physical Database Design and Tuning*. [S.l.]: CRC Press, 2001.

BRUNO, N.; CHAUDHURI, S. Interactive physical design tuning. In: *International Conference on Data Engineering*. [S.l.: s.n.], 2010. p. 1161–1164.

BRUNO, N.; CHAUDHURI, S.; RAMAMURTHY, R. Power hints for query optimization. In: *ICDE '09 Proceedings of the 2009 IEEE International Conference on Data*. Washington, DC, USA: IEEE Computer Society, 2009. p. 469–480. ISBN 978-0-7695-3545-6. Disponível em: <http://dx.doi.org/10.1109/ICDE.2009.68>.

COUNCIL, T. P. P. Tpc. 2007. Disponível em http://www.tpc.org.

DAGEVILLE, B.; DIAS, K. Oracle's self-tuning architecture and solutions. In: *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*. Oracle, USA: IEEE Computer Society, 2006.

ELLIOTT, B. et al. A complete translation from sparql into efficient sql. In: *IDEAS '09 Proceedings of the 2009 International Database Engineering & Applications Symposium*. New York, USA: [s.n.], 2009.

ELMASRI, R.; NAVATHE, S. B. *Sistemas de Banco de Dados: Fundamentos e Aplicações*. [S.l.]: Addison Wesley, 2005.

ELMASRI, R.; NAVATHE, S. B. *Sistemas de Banco de Dados: Fundamentos e Aplicações*. [S.l.]: earson Education, 2005.

HERODOTOU, H.; BABU, S. Automated sql tuning through trial and (sometimes) error. In: *DBTest '09 Proceedings of the Second International Workshop on Testing Database Systems*. New York, NY, USA: ACM, 2009. Disponível em: <http://dx.doi.org/10.1145/1594156.1594171>.

KODAGANALLUR, V. Incorporating language processing into java applications: a javacc tutorial. *Software, IEEE*, IEEE, Stillman Sch. of Bus., Seton Hall Univ., South Orange, NJ, USA, v. 21, p. 70–77, July 2004. ISSN 0740-7459.

KRISHNAPRASAD, M. et al. Query rewrite for xml in oracle xml db. In: *VLDB '04 Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*. Toronto, Canada: VLDB Endowment, 2004. p. 1134 – 1145. ISBN 0-12-088469-0.

LEMOS, L.; HOLANDA, P.; MONTEIRO, J. M. Um framework para a avaliaçãoo de desempenho de bancos dedados. In: *Sessão de Demos, Simpósio Brasileiro de Bancos de Dados - SBBD*. Santa Catarina, RS: [s.n.], 2011.

LIFSCHITZ, S.; MILANES, A. Y.; SALLES, M. A. V. *State of The Art in Self-Tuning Relational Database Systems (in portuguese)*. [S.l.], 2004.

LIFSCHITZ, S.; MILANéS, A. Y.; SALLES, M. A. V. Estudo da arte em auto-sintonia de sgbd relacionais. In: *Monografias em ciência da computação, Departamento de Informática*. Rio de Janeiro, Brasil: [s.n.], 2004.

MAIER, C. et al. Parinda: an interactive physical designer for postgresql. In: *Proceedings of the 13th International Conference on Extending Database Technology*. New York, NY, USA: ACM, 2010. (EDBT '10), p. 701–704. ISBN 978-1-60558-945-9. Disponível em: <http://doi.acm.org/10.1145/1739041.1739131>.

MAIER, C. et al. Parinda: An interactive physical designer for postgresql. In: *EDBT '10 Proceedings of the 13th International Conference on Extending Database Technology*. New York, USA: [s.n.], 2010.

MERLO, E.; LETARTE, D.; ANTONIOL, G. Automated protection of php applications against sql-injection attacks. In: *CSMR '07 Proceedings of the 11th European Conference on Software Maintenance and Reengineering*. [S.l.: s.n.], 2007.

MONTEIRO, J. M.; LIFSCHITZ, S.; BRAYNER, A. An architecture for automated index tuning. In: *N PH.D. AND M.S. WORKSHOP OF THE BRAZILIAN SYMPOSIUM ON DATABASE*. [S.l.: s.n.], 2006.

MONTEIRO, J. M.; LIFSCHITZ, S.; BRAYNER, A. An architecture for automated index tuning. In: *Proceedings of the Brazilian Symposium on Databases, Ph.D. session*. [S.l.: s.n.], 2006.

MORELLI, E. M. T. *Automatic Index Recreation in a Relational DBMS (in portuguese)*. Tese (Doutorado) — Department of Informatics, PUC-Rio, 2006.

OPTIMIZER, E. D. *Embarcadero DB Optimizer XE*. 2010. Available: http://www.embarcadero.com/products/db-optimizer-xe. June 2011.

QUEST, S. O. f. O. *Quest SQL Optimizer for Oracle*. 2010. Available: http://www.quest.com/SQL-Optimizer-for-Oracle. June 2011.

RAMAKRISHNAN, R.; GEHRKE, J. *Sistemas de Gerenciamento de Banco de Dados*. [S.l.]: McGraw Hill, 2008. ISBN 9788577260270.

RAMALHO, J. A. *Oracle 9i*. [S.l.]: Berkeley Brasil, 2002.

SALLES, M. A. V. *Autonomic index creation in databases (in portuguese)*. Tese (Doutorado) — Department of Informatics, PUC-Rio, 2004.

SCHNAITTER, K.; POLYZOTIS, N. Semi-automatic index tuning: Keeping dbas in the loop. *Proceedings of the VLDB Endowment*, v. 5, n. 5, p. 478–489, 2012.

SHASHA, D.; BONNET, P. *Database Tuning: Principles, Experiments and Troubleshooting Techniques*. [S.l.]: Morgan Kaufmann, 2003. ISBN 978-1558607538.

STUDIO, I. O. D. *IBM Optim Development Studio*. 2010. Available: http://www-01.ibm.com/software/data/optim/development-studio. June 2011.

THURAISINGHAM, B. et al. Um framework para a avaliaçãoo de desempenho de bancos dedados. In: *Networking, Applications and Worksharing (CollaborateCom), 2010 6th International Conference on*. Texas, USA: [s.n.], 2010.

VIEIRA, M.; DURãES, J.; MADEIRA, H. Especificação e validação de benchmarks de confiabilidade para sistemas transacionais. In: *IEEE Latin America Transactions*. [S.l.: s.n.], 2005.

WEIKUM, G. et al. Self-tuning database technology and information services: from wishful thinking to viable engineering. In: *Proceedings of the International Conference on Very Large Data Bases*. [S.l.: s.n.], 2002. p. 20–31.

APÊNDICE A – RESULTADOS DOS TESTES REALIZADOS COM A FERRAMENTA QUEST SQL OPTIMIZER FOR ORACLE UTILIZANDO O *BENCHMARK* TPC-H

A Quest SQL Optimizer for Oracle é uma ferramenta capaz de identificar oportunidades de sintonia em instruções SQL. Essa ferramenta utiliza a reescrita de instruções SQL e a aplicação de *hints* para gerar várias instruções semanticamente equivalentes a uma instrução SQL. Após produzir essas instruções, a ferramenta calcula o custo de execução de cada uma delas e as executa para obter o tempo de execução de cada uma delas. Assim, o DBA pode comparar o desempenho da instrução SQL original com o desempenho das instruções geradas pela ferramenta.

Com essa ferramenta os DBAs também podem gerenciar o desempenho da intrução SQL após alterações no banco de dados, tais como a criação do índice, alterações de configuração, *upgrades*, migrações e outros. Os DBAs podem analisar e comparar planos de execução de várias instruções SQL em ambientes de banco de dados diferentes para identificar variações e degradação de desempenho.

Para testar a eficácia da ferramenta, realizamos testes utilizando o *bechmark* TPC-H. A carga de trabalho utilizada nos testes foi composta por 21 consultas pertencentes ao *benchmark* TPC-H. O ambiente de experimentação utilizado foi composto por uma estação Core i3-2100 3.10GHz, com 4GB de Ram e 500 GB de HD, e com o sistemas operacional Windows Seven. Nos testes, a ferramenta conseguiu ajustar as consultas 2, 3, 11, 12 e 20 utilizando a aplicação de *hints*. A seguir estão as 23 consultas TPC-H e a descrição dos resultados obtidos nos testes realizados.

1. Consulta 01:

A ferramenta não conseguiu produzir nenhuma consulta semanticamente equivalente a consulta 01 da base TPH-C e gatou 1 segundo tentar ajustá-la. A seguir está a consulta 01.

•Consulta 01 orginal (Tempo de execução = 8.210 ms):

```
SELECT l_returnflag, l_linestatus, sum(l_quantity) as sum_qty, sum(l_extendedprice) as
        sum_base_price, sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
        sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
        avg(l_quantity) as avg_qty, avg(l_extendedprice) as avg_price,
        avg(l_discount) as avg_disc, count(*) as count_order
```

```
FROM lineitem
WHERE l_shipdate <= to_date('30/11/1998','DD/MM/YYYY')
GROUP BY l_returnflag, l_linestatus
ORDER BY l_returnflag, l_linestatus
```

2. Consulta 02:

A ferramenta encontrou 115 consultas semanticamente equivalentes as consulta 02, das quais 3 apresentaram os custo do plano de execução e tempo de execução menores que os da consulta 02. A ferramenta precisou de 116 segundos para produzir as 115 consultas. A seguir estão a consulta 02 e a consulta com menor tempo de execução gerada pela ferramenta a partir da consulta 02.

•Consulta 02 original (Tempo de execução = 310ms):

```
SELECT s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_comment
FROM part, supplier, partsupp, nation, region
WHERE p_partkey = ps_partkey And s_suppkey = ps_suppkey And p_size = 20 And
      p_type Like '%COPPER' And s_nationkey = n_nationkey And n_regionkey =
      r_regionkey And r_name = 'AMERICA' And
      ps_supplycost = (SELECT min(ps_supplycost)
                       FROM partsupp, supplier, nation, region
                       WHERE p_partkey = ps_partkey And s_suppkey = ps_suppkey
                       And s_nationkey = n_nationkey And n_regionkey = r_regionkey
                       And r_name = 'AMERICA')
ORDER BY s_acctbal desc, n_name, s_name, p_partkey
```

•Consulta 02 após ser ajustada pela ferramenta (Tempo de execução = 128ms):

```
SELECT /*+ NO_CPU_COSTING */ s_acctbal, s_name, n_name, p_partkey, p_mfgr,
       s_address, s_phone, s_comment
FROM part, supplier SUPPLIER1, partsupp PARTSUPP1, nation NATION1,
     region REGION1
WHERE p_partkey = ps_partkey And s_suppkey = ps_suppkey And p_size = 20
      And p_type Like '%COPPER' And s_nationkey = n_nationkey And
      n_regionkey = r_regionkey And r_name = 'AMERICA' And
      ps_supplycost = (SELECT min(ps_supplycost)
                       FROM partsupp PARTSUPP2, supplier SUPPLIER2,
                            nation NATION2, region REGION2
                       WHERE p_partkey = ps_partkey And s_suppkey = ps_suppkey
                            And s_nationkey = n_nationkey And regionkey =
                            r_regionkey And r_name = 'AMERICA')
ORDER BY s_acctbal desc, n_name, s_name, p_partkey
```

3. Consulta 03:

A ferramenta encontrou 53 consultas equivalentes a consulta 03, das quais 3 apresentaram o custo do plano de execução menor que o da original e apenas 1 apresentou o tempo de execução menor que o da original. A ferramenta gastou 58 segundos para produzir as 53 consultas. A seguir estão a consulta 03 e a consulta com menor tempo de execução gerada pela ferramenta a partir da consulta 03.

•Consulta 03 original (Tempo de execução = 6.393ms):

```
SELECT l_orderkey, sum(l_extendedprice * (1 - l_discount)) as revenue,
        o_orderdate, o_shippriority FROM customer, orders, lineitem
WHERE c_mktsegment = 'AUTOMOBILE' And c_custkey = o_custkey And
        l_orderkey = o_orderkey And o_orderdate ¡ to_date('31/12/1998','DD/MM/YYYY')
        And l_shipdate > to_date('01/01/1991','DD/MM/YYYY')
GROUP BY l_orderkey, o_orderdate, o_shippriority
ORDER BY revenue desc, o_orderdate
```

•Consulta 03 após ser ajustada pela ferramenta (Tempo de execução = 4.900ms):

```
SELECT l_orderkey, sum(l_extendedprice *(1 - l_discount)) as revenue, o_orderdate,
        o_shippriority
FROM customer, orders, lineitem
WHERE c_mktsegment = NVL('AUTOMOBILE', UID) And c_custkey = o_custkey And
        l_orderkey =o_orderkey And o_orderdate < to_date('31/12/1998', 'DD/MM/YYYY')
        And l_shipdate > to_date('01/01/1991', 'DD/MM/YYYY')
GROUP BY l_orderkey, o_orderdate, o_shippriority
ORDER BY revenue desc, o_orderdate
```

4. Consulta 04:

A ferramenta encontrou 14 consultas semanticamente equivalentes a consulta 04, das quais apenas 1 apresentou o custo do plano de execução menor que o da original, mas não possuiu o tempo de execução menor. A ferramenta precisou de 15 segundos para produzir as 14 consultas. A seguir estão a consulta 04 a consulta com menor tempo de execução gerada pela ferramenta a partir da consulta 04.

•Consulta 04 original (Tempo de execução = 4.204ms):

```
SELECT o_orderpriority, count(*) as order_count
FROM orders
WHERE o_orderdate >= to_date('01/08/1998','DD/MM/YYYY') And
        o_orderdate < to_date('08/11/1998','DD/MM/YYYY') And
        exists (SELECT *
                FROM lineitem
                WHERE l_orderkey = o_orderkey And l_commitdate < l_receiptdate )
GROUP BY o_orderpriority ORDER BY o_orderpriority
```

•Consulta 04 após ser sintonizada pela ferramenta (Tempo de execução = mais de 1 min):

```
SELECT o_orderpriority, count(*) as order_count
```

```
FROM orders
WHERE o_orderdate >= to_date('01/08/1998', 'DD/MM/YYYY') And
        o_orderdate < to_date('08/11/1998', 'DD/MM/YYYY') And
        o_orderkey IN (SELECT NVL(l_orderkey, l_orderkey)
                FROM lineitem
                WHERE l_commitdate < l_receiptdate)
GROUP BY o_orderpriority
ORDER BY o_orderpriority
```

5. Consulta 05:

A ferramenta encontrou 228 consultas semanticamente equivalentes a consulta 05, das quais 5 apresentaram o custo do plano de execução menor que o da original, mas nenhuma apresentou o tempo de execução menor. A ferramenta precisou de 229 segundos para produzir as 228 consultas. A seguir estão a consulta 05 e a consulta com menor tempo de execução gerada pela ferramenta a partir da consulta 05.

•Consulta 05 original (Tempo de execução = 4.658ms):

```
SELECT n_name, sum(l_extendedprice * (1 - l_discount)) as revenue
FROM customer, orders, lineitem, supplier, nation, region
WHERE c_custkey = o_custkey And l_orderkey = o_orderkey And
        l_suppkey = s_suppkey And c_nationkey = s_nationkey And
        s_nationkey = n_nationkey And n_regionkey = r_regionkey
        And r_name = 'AMERICA' And o_orderdate >= to_date('01/08/1991','DD/MM/YYYY')
        And o_orderdate < to_date('01/08/1992','DD/MM/YYYY')
GROUP BY n_name
ORDER BY revenue desc
```

•Consulta 05 após ser sintonizada pela ferramenta (Tempo de execução = 4.673ms):

```
SELECT /*+ USE_NL(REGION,NATION) */ n_name,
        sum(l_extendedprice * (1 - l_discount)) as revenue
FROM customer, orders, lineitem, supplier, nation, region
WHERE c_custkey = o_custkey And l_orderkey = o_orderkey And
        l_suppkey = s_suppkey And c_nationkey = s_nationkey And
        s_nationkey = n_nationkey And n_regionkey = r_regionkey
        And r_name = 'AMERICA' And o_orderdate >= to_date('01/08/1991','DD/MM/YYYY')
        And o_orderdate < to_date('01/08/1992','DD/MM/YYYY')
GROUP BY n_name
ORDER BY revenue desc
```

6. Consulta 06:

A ferramenta encontrou apenas 1 consulta semanticamente equivalente a consulta 06, entretanto o custo do seu plano de execução não era menor que o da original. A seguir está a consulta 06:

•Consulta 06 original (Tempo de execução = 2.313ms):

```
SELECT sum(l_extendedprice * l_discount) as revenue
FROM lineitem
WHERE l_shipdate >= to_date('07/01/1998','DD/MM/YYYY') And
        l_shipdate < to_date('07/01/1999','DD/MM/YYYY')
        And l_discount between 2- 0.01 And 2 + 0.01 And l_quantity < 5
```

7. onsulta 07:

A ferramenta encontrou 110 consultas semanticamente equivalentes a consulta 07, das quais apenas uma apresentou o custo do plano menor que o da original, mas não apresentou o tempo de execução menor do que o da original. A ferramenta precisou de 111 segundos para produzir as 110 consultas. A seguir, estão a consulta 07 e a consulta com menor tempo de execução gerada pela ferramenta a partir da consulta 07.

•Consulta 07 original (Tempo de execução = 4.658ms):

```
SELECT supp_nation, cust_nation, l_year, sum(volume) as revenue
FROM (SELECT n1.n_name as supp_nation, n2.n_name as cust_nation,
                extract(year from l_shipdate) as l_year,
                l_extendedprice * (1 - l_discount) as volume
        FROM supplier, lineitem, orders, customer, nation n1, nation n2
        WHERE s_suppkey = l_suppkey And o_orderkey = l_orderkey And
                c_custkey = o_custkey And s_nationkey = n1.n_nationkey And
                c_nationkey = n2.n_nationkey And ( (n1.n_name = 'ARGENTINA' And
                n2.n_name = 'ARGENTINA') Or (n1.n_name = 'BRAZIL' And
                n2.n_name = 'BRAZIL') ) And l_shipdate between
                to_date('01/01/1995','DD/MM/YYYY') And
                to_date('31/12/1996','DD/MM/YYYY') )
GROUP BY supp_nation, cust_nation, l_year
ORDER BY supp_nation, cust_nation, l_year
```

•Consulta 07 após ser sintonizada pela ferramenta: (Tempo de execução = 7.808ms):

```
SELECT supp_nation, cust_nation, l_year, sum(volume) as revenue
FROM (SELECT n1.n_name as supp_nation, n2.n_name as cust_nation,
                extract(year from l_shipdate) as l_year,
                l_extendedprice * (1 - l_discount) as volume
        FROM supplier, lineitem, orders, customer, nation n1, nation n2
        WHERE s_suppkey = l_suppkey And o_orderkey = l_orderkey And
                c_custkey = o_custkey And s_nationkey = n1.n_nationkey And
                c_nationkey = n2.n_nationkey And ( (n1.n_name = 'ARGENTINA' And
                n2.n_name = 'ARGENTINA') Or (n1.n_name = 'BRAZIL' And
                n2.n_name = 'BRAZIL') ) And l_shipdate between
                to_date('01/01/1995','DD/MM/YYYY') And
                to_date('31/12/1996','DD/MM/YYYY') )
```

```
GROUP BY supp_nation, cust_nation, l_year
ORDER BY supp_nation, cust_nation, l_year
```

8. Consulta 08:

A ferramenta encontrou 140 consultas semanticamente equivalentes a consulta 08, das quais apenas uma apresentou o custo do plano menor que o da original, mas não apresentou o tempo de execução menor. A ferramenta precisou de 141 segundos para produzir as 140 consultas. A seguir estão a consulta 08 e a consulta com menor tempo de execução gerada pela ferramenta a partir da consulta 08.

•Consulta 08 original (Tempo de execução = 4.567ms):

```
SELECT o_year, sum(case when nation = 'UNITED STATES' then volume else 0 end) /
        sum(volume) as mkt_share
FROM ( SELECT extract(year from o_orderdate) as o_year, l_extendedprice * (1 - l_discount)
            as volume, n2.n_name as nation
        FROM part, supplier, lineitem, orders, customer, nation n1, nation n2, region
WHERE p_partkey = l_partkey And s_suppkey = l_suppkey And l_orderkey = o_orderkey
        And o_custkey = c_custkey And c_nationkey = n1.n_nationkey And
        n1.n_regionkey = r_regionkey And r_name = 'AFRICA' And s_nationkey =
        n2.n_nationkey And o_orderdate between to_date('01/01/1995','DD/MM/YYYY')
        And to_date('31/12/1996','DD/MM/YYYY') And
        p_type = 'ECONOMY BRUSHED COPPER')
GROUP BY o_year
ORDER BY o_year
```

•Consulta 08 após ser sintonizada pela ferramenta (Tempo de execução = 4.770ms):

```
SELECT o_year, sum(case when nation = 'UNITED STATES' then volume else 0 end) /
        sum(volume) as mkt_share
FROM ( SELECT extract(year from o_orderdate) as o_year, l_extendedprice * (1 - l_discount)
            as volume, n2.n_name as nation
        FROM part, supplier, lineitem, orders, customer, nation n1, nation n2, region
WHERE p_partkey = l_partkey And s_suppkey = l_suppkey And l_orderkey = o_orderkey
        And o_custkey = c_custkey And c_nationkey = n1.n_nationkey And
        n1.n_regionkey = r_regionkey And r_name = 'AFRICA' And s_nationkey =
        n2.n_nationkey And o_orderdate between to_date('01/01/1995','DD/MM/YYYY')
        And to_date('31/12/1996','DD/MM/YYYY') And
        p_type = NVL('ECONOMY BRUSHED COPPER', UID))
GROUP BY o_year
ORDER BY o_year
```

9. Consulta 09:

A ferramenta encontrou 191 consultas equivalentes a consulta 09, das quais 5 apresentaram o custo do plano de execução menor que o da original e nenhuma apresentou o tempo de execução

menor que o da original. A ferramenta precisou de 192 segundos para produzir as 191 consultas. A seguir estão a consulta 09 e a consulta com menor tempo de execução gerada pela ferramenta a partir da consulta 09.

•Consulta 09 original (Tempo de execução = 6.181ms):

```
SELECT nation, o_year, sum(amount) as sum_profit
FROM ( SELECT n_name as nation, extract(year from o_orderdate) as o_year,
        l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
       FROM part, supplier, lineitem, partsupp, orders, nation
       WHERE s_suppkey = l_suppkey And ps_suppkey = l_suppkey And
             ps_partkey = l_partkey And p_partkey = l_partkey And
             o_orderkey = l_orderkey And s_nationkey = n_nationkey
             And p_name like '%blush%')
GROUP BY nation, o_year
ORDER BY nation, o_year desc
```

•Consulta 09 original (Tempo de execução = 6.749ms):

```
SELECT nation, o_year, sum(amount) as sum_profit
FROM ( SELECT /*+ USE_MERGE(PART,SUPPLIER) */ n_name as nation,
        extract(year from o_orderdate) as o_year,
        l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
       FROM part, supplier, lineitem, partsupp, orders, nation
       WHERE s_suppkey = l_suppkey And ps_suppkey = l_suppkey And
             ps_partkey = l_partkey And p_partkey = l_partkey And
             o_orderkey = l_orderkey And s_nationkey = n_nationkey
             And p_name like '%blush%')
GROUP BY nation, o_year
ORDER BY nation, o_year desc
```

10. Consulta 10:

A ferramenta encontrou 118 consultas semanticamente equivalentes as consulta 10, das quais nenhuma apresentou o custo do plano de execução menor que o da consulta 10. A ferramenta precisou de 120 segundos para produzir as 118 consultas. A seguir está a consulta 10:

•Consulta 10 original (Tempo de execução = 3.160ms):

```
SELECT c_custkey, c_name, sum(l_extendedprice * (1 - l_discount)) as revenue, c_acctbal,
        n_name, c_address, c_phone, c_comment
FROM customer, orders, lineitem, nation
WHERE c_custkey = o_custkey And l_orderkey = o_orderkey And
        o_orderdate >= to_date('01/08/1992','DD/MM/YYYY') And
        o_orderdate ¡ to_date('01/11/1992','DD/MM/YYYY') And
        l_returnflag = 'R' And c_nationkey = n_nationkey
GROUP BY c_custkey, c_name, c_acctbal, c_phone, n_name, c_address, c_comment
ORDER BY revenue desc
```

11. Consulta 11:

A ferramenta encontrou 139 consultas semanticamente equivalentes a consulta 11, das quais 8 apresentaram o custo do plano de execução menor que o da original e 4 apresentaram o tempo de execução menor que o da original. A ferramenta precisou de 140 segundos para produzir as 139 consultas. A seguir estão a consulta 11 e a consulta com menor tempo de execução gerada pela ferramenta a partir da consulta 11.

•Consulta 11 original (Tempo de execução = 600ms):

```
SELECT ps_partkey, sum(ps_supplycost * ps_availqty) as value
FROM partsupp, supplier, nation
WHERE ps_suppkey = s_suppkey And s_nationkey = n_nationkey
        And n_name = 'BRAZIL'
GROUP BY ps_partkey
HAVING sum(ps_supplycost * ps_availqty) > (SELECT sum(ps_supplycost * ps_availqty)*2
                                           FROM partsupp, supplier, nation
                                           WHERE ps_suppkey = s_suppkey And
                                           s_nationkey = n_nationkey And
                                           n_name = 'BRAZIL')
ORDER BY value desc
```

•Consulta 11 após ser sintonizada pela ferramenta (Tempo de execução = 259ms):

```
SELECT /*+ NO_CPU_COSTING */ ps_partkey,
        sum(ps_supplycost * ps_availqty) as value
FROM partsupp PARTSUPP1, supplier SUPPLIER1, nation NATION1
WHERE ps_suppkey = s_suppkey And s_nationkey = n_nationkey
        And n_name = 'BRAZIL'
GROUP BY ps_partkey
HAVING sum(ps_supplycost * ps_availqty) > (SELECT sum(ps_supplycost * ps_availqty)*2
                                           FROM partsupp PARTSUPP2,
                                                supplier SUPPLIER2,
                                                nation NATION2
                                          WHERE ps_suppkey = s_suppkey And
                                                s_nationkey = n_nationkey And
                                                n_name = 'BRAZIL')
ORDER BY value desc
```

12. Consulta 12:

A ferramenta encontrou 20 consultas semanticamente equivalentes a consulta 12, das quais apenas uma apresentou os custo do plano de execução e tempo de execução menores que os da original. A ferramenta precisou de 21 segundos para produzir as 20 consultas. A seguir estão a consulta 12 e a consulta com menor tempo de execução gerada pela ferramenta a partir da consulta 12.

•Consulta 12 original (Tempo de execução = 1.995ms):

```
SELECT l_shipmode, sum(case when o_orderpriority = '1-URGENT' or
        o_orderpriority = '2-HIGH' then 1 else 0 end) as high_line_count,
        sum(case when o_orderpriority <> '1-URGENT' and
        o_orderpriority <> '2-HIGH' then 1 else 0 end) as low_line_count
FROM orders, lineitem
WHERE o_orderkey = l_orderkey And l_shipmode in ('TRUCK', 'AIR') And
        l_commitdate < l_receiptdate And l_shipdate < l_commitdate And
        l_receiptdate >= to_date('01/01/1996','DD/MM/YYYY') And
        l_receiptdate < to_date('01/01/1997','DD/MM/YYYY')
GROUP BY l_shipmode
ORDER BY l_shipmode
```

•Consulta 12 após ser sintonizada pela ferramenta (Tempo de execução = 1.993ms):

```
SELECT /*+ ORDERED */ l_shipmode, sum(case when o_orderpriority = '1-URGENT'
        or o_orderpriority = '2-HIGH' then 1 else 0 end) as high_line_count,
        sum(case when o_orderpriority <> '1-URGENT' and
        o_orderpriority <> '2-HIGH' then 1 else 0 end) as low_line_count
FROM orders, lineitem
WHERE o_orderkey = l_orderkey And l_shipmode in ('TRUCK', 'AIR') And
        l_commitdate < l_receiptdate And l_shipdate < l_commitdate And
        l_receiptdate >= to_date('01/01/1996','DD/MM/YYYY') And
        l_receiptdate < to_date('01/01/1997','DD/MM/YYYY')
GROUP BY l_shipmode
ORDER BY l_shipmode
```

13. Consulta 13:

A ferramenta encontrou 2 consultas semanticamente equivalentes as consulta 13, das quais nenhuma apresentou o custo do plano de execução menor que o da consulta 13. A ferramenta precisou de 3 segundos para produzir as 2 consultas. A seguir está a consulta 13:

•Consulta 13 original (Tempo de execução = 1.995ms):

```
SELECT c_count, count(*) as custdist
FROM( SELECT c_custkey, count(o_orderkey) as c_count
        FROM customer left outer join orders on c_custkey = o_custkey And
                o_comment not like '%even%deposits%'
        GROUP BY c_custkey )
GROUP BY c_count
ORDER BY custdist desc, c_count desc
```

14. Consulta 14:

A ferramenta encontrou 5 consultas equivalentes a consulta 14, das quais apenas uma apresentou o custo do plano de execução menor que o da original, mas não apresentou o tempo de

execução menor que o da original. A ferramenta precisou de 6 segundos para produzir as 5 consultas. A seguir estão a consulta 14 e a consulta com menor tempo de execução gerada pela ferramenta a partir da consulta 14.

•Consulta 14 original (Tempo de execução = 512 ms):

```
SELECT 100.00 * sum(case when p_type like 'PROMO%' then
            l_extendedprice * (1 - l_discount) else 0 end) /
            sum(l_extendedprice * (1 - l_discount)) as promo_revenue
FROM lineitem, part
WHERE l_partkey = p_partkey And l_shipdate >= to_date('01/02/1996','DD/MM/YYYY')
          And l_shipdate < to_date('01/03/1996','DD/MM/YYYY')
```

•Consulta 14 após ser sintonizada pela ferramenta (Tempo de execução = 538 ms):

```
SELECT /*+ ORDERED */ 100.00 * sum(case when p_type like 'PROMO%' then
            l_extendedprice * (1 - l_discount) else 0 end) /
            sum(l_extendedprice * (1 - l_discount)) as promo_revenue
FROM lineitem, part
WHERE l_partkey = p_partkey And l_shipdate >= to_date('01/02/1996','DD/MM/YYYY')
          And l_shipdate < to_date('01/03/1996','DD/MM/YYYY')
```

15. Consulta 16:

A ferramenta encontrou 35 consultas semanticamente equivalentes a consulta 16, das quais 2 apresentaram o custo do plano de execução menor que o da original mas nenhuma apresentou o tempo de execução menor que o da original. A ferramenta precisou de 36 segundos para produzir as 35 consultas. A seguir estão a consulta 16 e a consulta com menor tempo de execução gerada pela ferramenta.

•Consulta 16 original (Tempo de execução = 278.244ms):

```
SELECT p_brand, p_type, p_size, count(distinct ps_suppkey) as supplier_cnt
FROM partsupp, part
WHERE p_partkey = ps_partkey And p_brand <> 'Brand#13' And p_type
            not like 'STANDARD%' And p_size in (7, 12, 14, 16, 21, 23, 32, 43) And
            ps_suppkey not in (SELECT s_suppkey
                                    FROM supplier
                                    WHERE s_comment like '%Customer%Complaints%' )
GROUP BY p_brand, p_type, p_size
ORDER BY supplier_cnt desc, p_brand, p_type, p_size
```

•Consulta 16 após ser sintonizada pela ferramenta (Tempo de execução = 280.190ms):

```
SELECT /*+ ORDERED */p_brand, p_type, p_size,
            count(distinct ps_suppkey) as supplier_cnt
FROM partsupp, part
WHERE p_partkey = ps_partkey And p_brand <> 'Brand#13' And p_type
```

```
        not like 'STANDARD%' And p_size in (7, 12, 14, 16, 21, 23, 32, 43) And
        ps_suppkey not in (SELECT s_suppkey
                           FROM supplier
                           WHERE s_comment like '%Customer%Complaints%' )
GROUP BY p_brand, p_type, p_size
ORDER BY supplier_cnt desc, p_brand, p_type, p_size
```

16. Consulta 17:

A ferramenta encontrou 29 consultas semanticamente equivalentes as consulta 17, das quais nenhuma apresentou o custo do plano de execução menor que o da consulta 17. A ferramenta precisou de 30 segundos para produzir as 29 consultas. A seguir está a consulta 17.

•Consulta 17 original (Tempo de execução = 1.635 ms):

```
SELECT sum(l_extendedprice) / 7.0 as avg_yearly
FROM lineitem, part
WHERE p_partkey = l_partkey And p_brand = 'Brand#13' And
        p_container = 'JUMBO PKG'
        And l_quantity < (SELECT 0.2 * avg(l_quantity)
                          FROM lineitem
                          WHERE l_partkey = p_partkey )
```

17. Consulta 18:

A ferramenta encontrou 160 consultas semanticamente equivalentes a consulta 18, das quais apenas uma apresentou o custo do plano de execução menor que o da original, mas não apresentou o tempo de execução menor que o da original. A ferramenta precisou de 161 segundos para produzir as 160 consultas. A seguir estão a consulta 18 e a consulta com menor tempo de execução gerada pela ferramenta a partir consulta 18.

•Consulta 18 original (Tempo de execução = 17.384ms):

```
SELECT c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice, sum(l_quantity)
FROM customer, orders, lineitem
WHERE o_orderkey in ( SELECT l_orderkey
                      FROM lineitem
                      GROUP BY l_orderkey
                      HAVING sum(l_quantity) > 3 )
        And c_custkey = o_custkey And o_orderkey = l_orderkey
GROUP BY c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice
ORDER BY o_totalprice desc, o_orderdate
```

•Consulta 18 original (Tempo de execução = 22.306ms):

```
SELECT /*+ NO_CPU_COSTING */ c_name, c_custkey, o_orderkey, o_orderdate,
        o_totalprice, sum(l_quantity)
```

```
FROM customer, orders, lineitem
WHERE o_orderkey in ( SELECT l_orderkey
                      FROM lineitem
                      GROUP BY l_orderkey
                      HAVING sum(l_quantity) > 3 )
      And c_custkey = o_custkey And o_orderkey = l_orderkey
GROUP BY c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice
ORDER BY o_totalprice desc, o_orderdate
```

18. Consulta 19:

A ferramenta encontrou 86 consultas semanticamente equivalentes as consulta 19, das quais nenhuma apresentou o custo do plano de execução menor que o da consulta 19. A ferramenta precisou de 87 segundos para produzir as 86 consultas. A seguir está a consulta 19:

•Consulta 19 original (Tempo de execução = 3.228 ms):

```
SELECT sum(l_extendedprice* (1 - l_discount)) as revenue
FROM lineitem, part
WHERE ( p_partkey = l_partkey And p_brand = 'Brand#13' And
        p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG') And l_quantity>=4
        And l_quantity <= 14 And p_size between 1 And 5 And
        l_shipmode in ('AIR', 'AIR REG') And l_shipinstruct =
        'DELIVER IN PERSON' ) Or ( p_partkey = l_partkey And p_brand = 'Brand#44'
        And p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
        And l_quantity >= 5 And l_quantity <= 15 And p_size between 1 And 10 And
        l_shipmode in ('AIR', 'AIR REG') And l_shipinstruct =
        'DELIVER IN PERSON' ) Or ( p_partkey = l_partkey And p_brand = 'Brand#53'
        And p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG') And
        l_quantity >= 6 And l_quantity <= 16 And p_size between 1 And 15 And
        l_shipmode in ('AIR', 'AIR REG') And l_shipinstruct = 'DELIVER IN PER-
SON' )
```

19. Consulta 20:

A ferramenta encontrou 92 consultas semanticamente equivalentes a consulta 20, das quais 8 apresentaram o custo do plano de execução menor que da original e 3 apresentaram o tempo de execução menor que a original. A ferramenta precisou de 93 segundos para produzir as 92 consultas. A seguir estão a consulta 20 e a consulta com menor tempo de execução gerada pela ferramenta a partir da consulta 20.

•Consulta 20 original (Tempo de execução = 3.261ms):

```
SELECT s_name, s_address
FROM supplier, nation
WHERE s_suppkey in ( SELECT distinct (ps_suppkey)
```

```
                    FROM partsupp, part
                    WHERE ps_partkey = p_partkey And p_name like 'dim%' And
                    ps_availqty > (SELECT 0.5 * sum(l_quantity)
                                FROM lineitem
                                WHERE l_partkey = ps_partkey And
                                        l_suppkey = ps_suppkey And
                            l_shipdate >= to_date('01/03/1997','DD/MM/YYYY') And
                            l_shipdate < to_date('01/03/1998','DD/MM/YYYY') ))
        And s_nationkey = n_nationkey And n_name = 'ARGENTINA'
ORDER BY s_name
```

•Consulta 20 original (Tempo de execução = 3.212ms):

```
SELECT /*+ FIRST_ROWS */ s_name, s_address
FROM supplier, nation
WHERE s_suppkey in ( SELECT distinct NVL(ps_suppkey)
                    FROM partsupp, part
                    WHERE ps_partkey = p_partkey And p_name like 'dim%' And
                    ps_availqty > (SELECT 0.5 * sum(l_quantity)
                                FROM lineitem
                                WHERE l_partkey = ps_partkey And
                                        l_suppkey = ps_suppkey And
                            l_shipdate >= to_date('01/03/1997','DD/MM/YYYY') And
                            l_shipdate < to_date('01/03/1998','DD/MM/YYYY') ))
        And s_nationkey = n_nationkey And n_name = 'ARGENTINA'
ORDER BY s_name
```

20. Consulta 21:

A ferramenta encontrou 120 consultas semanticamente equivalentes a consulta 21, das quais 5 apresentaram o custo do plano de execução menor que o da original, mas nenhuma apresentou o tempo de execução menor que o da original. A ferramenta precisou de 121 segundos para produzir as 120 consultas. A seguir estão a consulta 18 e a consulta com menor tempo de execução gerada pela ferramenta a partir da consulta 21.

•Consulta 21 original (Tempo de execução = 8.660 ms):

```
SELECT s_name, count(*) as numwait
FROM supplier, lineitem l1, orders, nation
WHERE s_suppkey = l1.l_suppkey And o_orderkey = l1.l_orderkey And
        o_orderstatus = 'F' And l1.l_receiptdate > l1.l_commitdate And
        exists (SELECT *
                FROM lineitem l2
               WHERE l2.l_orderkey=l1.l_orderkey And l2.l_suppkey<>l1.l_suppkey)
        And not exists (SELECT *
                        FROM lineitem l3
```

WHERE l3.l_orderkey = l1.l_orderkey And l3.l_suppkey<>
l1.l_suppkey And l3.l_receiptdate > l3.l_commitdate)
And s_nationkey = n_nationkey And n_name = 'BRAZIL'
GROUP BY s_name
ORDER BY numwait desc, s_name

•Consulta 21 após ser sintonizada pela ferramenta (Tempo de execução = 9.030 ms):
SELECT /*+ USE_NL(SUPPLIER) */s_name, count(*) as numwait
FROM supplier, lineitem l1, orders, nation
WHERE s_suppkey = l1.l_suppkey And o_orderkey = l1.l_orderkey And
o_orderstatus = NVL('F', UID) And l1.l_receiptdate > l1.l_commitdate And
exists (SELECT *
FROM lineitem l2
WHERE l2.l_orderkey=l1.l_orderkey And l2.l_suppkey<>l1.l_suppkey)
And not exists (SELECT *
FROM lineitem l3
WHERE l3.l_orderkey = l1.l_orderkey And l3.l_suppkey<>
l1.l_suppkey And l3.l_receiptdate > l3.l_commitdate)
And s_nationkey = n_nationkey And n_name = 'BRAZIL'
GROUP BY s_name
ORDER BY numwait desc, s_name

21. Consulta 22:

A ferramenta encontrou uma consulta equivalente a consulta 19, cujo custo do plano de execução foi maior que o da consulta 22. A ferramenta precisou de 1 segundo para produzir a consulta. A seguir está a consulta 22:

•Consulta 22 original (Tempo de execução = 407 ms):
SELECT cntrycode, count(*) as numcust, sum(c_acctbal) as totacctbal
FROM (SELECT substr(c_phone, 1, 2) as cntrycode, c_acctbal
FROM customer
WHERE substr(c_phone, 1, 2) in ('25', '11', '13', '14', '30', '23', '18') And
c_acctbal > (SELECT avg(c_acctbal)
FROM customer
WHERE c_acctbal > 0.00 And substr(c_phone, 1, 2)
in ('25', '11', '13', '14', '30', '23', '18'))
And not exists (SELECT *
FROM orders
WHERE o_custkey = c_custkey))
GROUP BYcntrycode
ORDER BYcntrycode

APÊNDICE B – BENCHMARK TPC-H

O TPC-H é um *benchmark* de suporte a decisões que consiste em um conjunto de consultas *ad-hoc* voltadas para os negócios e modificações de dados simultâneas. Tem por finalidade simular e avaliar o desempenho de um ambiente de *Data Warehouse*. *Data Warehouse* é um grande repositório de dados coletados de diversas fontes que se destina a gerar informações para o nível gerencial sendo fonte para tomadas de decisão.

Os testes do método TPC-H são realizados sob uma estrutura padrão composta por oito tabelas. Destas, seis são tabelas dimensionais (*Region*, *Nation*, *Supplier*, *Part*, *Customer* e *Partsupp*) e duas de fatos (*Orders* e *Lineitem*).

As tabelas de fato estão no centro do modelo dimensional (modelo para suporte à decisão), seus valores são usados para medir o desempenho do negócio. As tabelas de dimensão são áreas ou focos do negócio e fornecem um método geral de organizar a informação corporativa provendo múltiplas perspectivas dos dados. As tabelas de fatos armazenam os fatos ocorridos e as chaves para as características correspondentes nas tabelas dimensionais. As consultas ocorrem inicialmente nas tabelas de dimensão e depois nas tabelas de fatos, assegurando a precisão dos dados.

A Figura B.1 mostra o modelo dimensional do *benchmark* TPC-H. Já a Figura B.2 ilustra o modelo relacional do *benchmark* TPC-H.

Algumas observações devem ser ressaltadas, uma vez analisada a Figura B.2 (MORELLI, 2006):

- O número que aparece logo abaixo do nome da tabela representa sua cardinalidade. Esta pode ser fixa (tabelas region e nation), ou variável, onde multiplica-se uma constante por um *scale factor* determinado no momento da criação da base. Por exemplo, caso SF valha 3, haverá 450.000 *tuplas* na tabela de clientes (customer).

- Estão representadas cinco regiões (continentes), que congregam vinte e cinco nações (tabelas region e nation, respectivamente). Clientes e Fornecedores (tabelas supplier e customer) estão associados às nações. Enquanto os primeiros realizam pedidos de compras (tabela orders), os segundos fornecem componentes (tabela part) de itens de compra. Como um fornecedor pode oferecer vários itens e um item pode ser disponibilizado por vários fornecedores, existe uma tabela para registrar esta relação $N \times N$ (partsupp). Finalmente, a tabela mais volumosa do modelo (lineitem) associa itens de compra a pedidos.

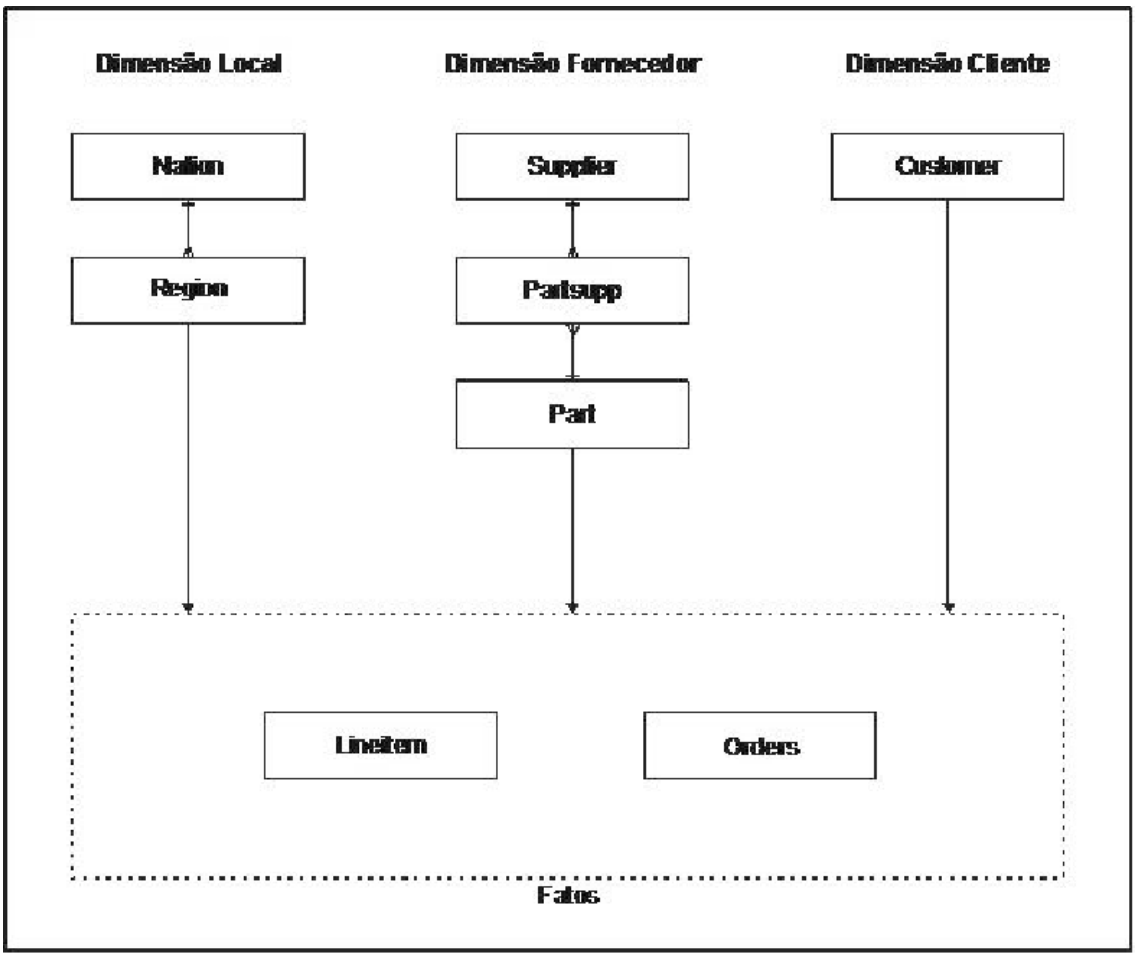

Figura B.1: Modelo Dimensional do TPC-H.

- Os parênteses ao lado dos nomes das tabelas indicam o prefixo utilizado para denominar os campos da tabela em questão. Desta forma, a chave primária da tabela de fornecedores chama-se S_SUPPKEY;
- As flechas indicam as associações entre chaves primárias e estrangeiras. Assim, vemos que a chave primária da tabela nation está vinculada ao campo nationkey na tabela de fornecedores (tabela supplier);

O tamanho total da base de dados depende do fator de escalabilidade (SF - *scale factor*). Para SF=1, a base completa ocupa aproximadamente 1 GB e os volumes de cada tabela são os que aparecem na Figura B.2. Já para SF=30, a base ocupará 30 GB e a tabela de itens de pedidos de compra possuirá cento e oitenta milhões de tuplas (30×6 milhões) (MORELLI, 2006).

A base de dados sofre acessos de um conjunto de consultas possuindo características *ad-hoc*, ou seja, não se conhece nem a ordem de execução, nem os parâmetros de cada uma das 22 consultas (leitura) e duas funções (uma insere dados e a outra elimina) (MORELLI, 2006).

O *benchmark* TPC-H compõe-se por vários testes, como revela a tabela B.1.

Antes de executar os testes, alimenta-se o banco de dados com base em arquivos texto previamente gerados. Isto acontece graças a uma ferramenta normalmente apelidada por *DBGEN*

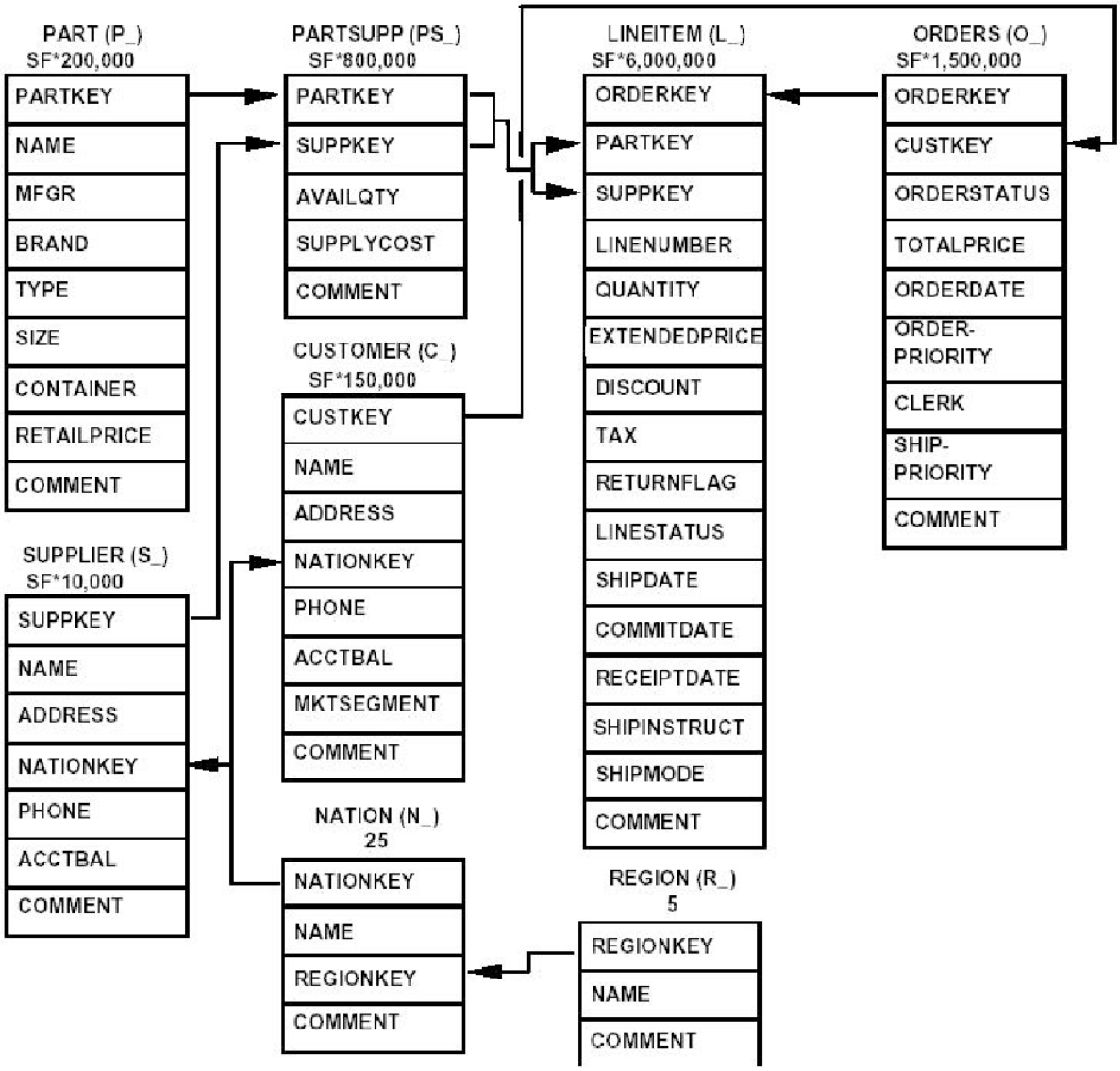

Figura B.2: Modelo Relacional do TPC-H.

Teste	Detalhes
Refresh Function 1 (RF1)	Insere *tuplas* nas duas maiores tabelas: orders e lineitem.
Refresh Function 2 (RF2)	Elimina tuplas das tabelas orders e lineitem.
Power	Compreendido pela *Refresh Function 1*, grupo completo de análises de consultas e *Refresh Function 2*.
Throughput	Dispara vários fluxos (*streams*) de comandos em paralelo. Invoca as vinte e duas consultas seguidas pelas *Refresh Functions*.
Performance	Reúne os testes *Power* e *Throughput*.

Tabela B.1: Testes que Compõem o *benchmark* TPC-H.

(*Database Generator*), seguindo preceitos de (COUNCIL, 2007). A ordem na qual são lidas as vinte e duas consultas, bem como seus parâmetros, também são gerados por um programa

SF	Fluxos
1	2
10	3
30	4
100	5
300	6

Tabela B.2: Quantidade de Fluxos Utilizados no Teste *Throughput*.

gerador denominado: *QGEN* (*Query Generator*).

A quantidade de fluxos de comandos utilizados no teste *Throughput* deve acompanhar o fator de escalabilidade da base de dados. A especificação oficial do *benchmark* TPC-H apresenta a seguinte correlação (Tabela B.2):

O teste *Throughput* produz duas medidas: a vazão (quantos comandos por segundo) e tempo transcorrido.

As funções de atualização (RF1 e RF2) trabalham sobre as maiores tabelas, orders e lineitem; a primeira função insere *tuplas* e a segunda exclui. Vale ressaltar que, como a execução de uma RF sempre vem seguida da outra, a quantidade de *tuplas* permanece estável, já que RF1 insere 0,1% de tuplas, enquanto RF2 elimina outras 0,1% de *tuplas* (MORELLI, 2006).

Tanto RF1 quanto RF2 recebem por argumento o fator de escalabilidade, coerente com o tamanho atual das tabelas. Assim, na tabela lineitem, quando SF=1, a execução de RF1 causará a inserção de seis mil tuplas: 6.000.000 $\times$ 0,01. Caso fosse necessário aumentar o percentual de *tuplas* afetadas, as funções poderiam ser executadas com argumento SF=30, o que, em bases com fator de escalabilidade um, teríamos o aumento trinta vezes maior (de 0,1% para 3% : 180.000 *tuplas*) (MORELLI, 2006).

B.1 O Ambiente OLAP

OLAP (*On-Line Analytical Processing*) é uma tecnologia que permite aos analistas de negócios, gerentes e executivos analisar e visualizar dados corporativos de forma rápida, consistente e principalmente interativa.

A funcionalidade OLAP é inicialmente caracterizada pela análise dinâmica e multidimensional dos dados consolidados de uma organização permitindo que as atividades do usuário final sejam tanto analíticas quanto navegacionais. A tecnologia OLAP é geralmente implementada em ambiente multiusuário e cliente/servidor, oferecendo assim respostas rápidas às consultas *ad-hoc*, não importando o tamanho do banco de dados nem sua complexidade. Hoje em dia, OLAP também vem sendo disponibilizada em ambiente *Web*.

A tecnologia OLAP auxilia o usuário a sintetizar informações corporativas por meio de visões comparativas e personalizadas, análises históricas, projeções e elaborações de cenários.

B.2 Cargas de Trabalho TPC-H

Como mencionado anteriormente, o *benchmark* TPC-H é composto por um conjunto de consultas *ad-hoc* que simulam as atividades encontradas em um ambiente OLAP. A seguir, apresentamos as consultas que compõem este *benchmark*:

```
1.  SELECT  l_returnflag,l_linestatus, sum(l_quantity) as sum_qty,
            sum(l_extendedprice) as sum_base_price,
            sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
            sum(l_extendedprice * (1 - l_discount) * (1 + l_tax))
                as sum_charge,
            avg(l_quantity) as avg_qty, avg(l_extendedprice) as avg_price,
            avg(l_discount) as avg_disc, count(*) as count_order
    FROM lineitem
    WHERE l_shipdate <= date'1998-12-01' - interval '10 days'
    GROUP BY l_returnflag, l_linestatus
    ORDER BY l_returnflag, l_linestatus


2.  SELECT  s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address,
            s_phone, s_comment
    FROM    part, supplier, partsupp, nation, region
    WHERE   p_partkey = ps_partkey and s_suppkey = ps_suppkey and p_size = 20
            and p_type like '%COPPER' and s_nationkey = n_nationkey
            and n_regionkey = r_regionkey and r_name = 'AMERICA'
            and ps_supplycost = ( SELECT min(ps_supplycost)
                                  FROM partsupp, supplier, nation, region
                                  WHERE p_partkey = ps_partkey
                                        and s_suppkey = ps_suppkey
                                        and s_nationkey = n_nationkey
                                        and n_regionkey = r_regionkey
                                        and r_name = 'AMERICA'
                                )
    ORDER BY s_acctbal desc, n_name, s_name, p_partkey


3.  SELECT  l_orderkey, sum(l_extendedprice * (1 - l_discount)) as revenue,
            o_orderdate, o_shippriority
    FROM    customer, orders, lineitem
    WHERE   c_mktsegment = 'AUTOMOBILE' and c_custkey = o_custkey
            and l_orderkey = o_orderkey and o_orderdate < date '19981231'
            and l_shipdate > date '19910101'
```

```
    GROUP BY l_orderkey, o_orderdate, o_shippriority
    ORDER BY revenue desc, o_orderdate
```

4.
```
    SELECT o_orderpriority, count(*) as order_count
    FROM   orders
    WHERE  o_orderdate >= date '19980801'
           and o_orderdate < date '19980808' + interval '3 month'
           and exists ( SELECT *
                        FROM lineitem
                        WHERE l_orderkey = o_orderkey
                              and l_commitdate < l_receiptdate
                      )
    GROUP BY o_orderpriority
    ORDER BY o_orderpriority
```

5.
```
    SELECT n_name, sum(l_extendedprice * (1 - l_discount)) as revenue
    FROM   customer, orders, lineitem, supplier, nation, region
    WHERE  c_custkey = o_custkey and l_orderkey = o_orderkey
           and l_suppkey = s_suppkey and c_nationkey = s_nationkey
           and s_nationkey = n_nationkey and n_regionkey = r_regionkey
           and r_name = 'AMERICA' and o_orderdate >= date '19910801'
           and o_orderdate < date '19910801' + interval '1 year'
    GROUP BY n_name
    ORDER BY revenue desc
```

6.
```
    SELECT sum(l_extendedprice * l_discount) as revenue
    FROM   lineitem
    WHERE  l_shipdate >= date '19980107'
           and l_shipdate < date '19980107' + interval '1 year'
           and l_discount between 2 - 0.01 and 2 + 0.01
           and l_quantity < 5
```

7.
```
    SELECT supp_nation, cust_nation, l_year, sum(volume) as revenue
    FROM ( SELECT n1.n_name as supp_nation, n2.n_name as cust_nation,
                  extract(year from l_shipdate) as l_year,
                  l_extendedprice * (1 - l_discount) as volume
           FROM   supplier, lineitem, orders, customer, nation n1, nation n2
           WHERE  s_suppkey = l_suppkey  and o_orderkey = l_orderkey
                  and c_custkey = o_custkey and s_nationkey = n1.n_nationkey
```

```
                and c_nationkey = n2.n_nationkey
                and (
                       (n1.n_name = 'ARGENTINA' and n2.n_name = 'ARGENTINA')
                       or
                       (n1.n_name = 'BRAZIL' and n2.n_name = 'BRAZIL')
                    )
                and l_shipdate between date '1995-01-01'
                and date '1996-12-31'
     ) as shipping
GROUP BY supp_nation, cust_nation, l_year
ORDER BY supp_nation, cust_nation, l_year
```

8.
```
SELECT o_year, sum(case when nation = 'UNITED STATES'
                    then volume else 0 end) / sum(volume) as mkt_share
FROM ( SELECT extract(year from o_orderdate) as o_year,
              l_extendedprice * (1 - l_discount) as volume,
              n2.n_name as nation
       FROM   part, supplier, lineitem, orders, customer,
              nation n1, nation n2, region
       WHERE  p_partkey = l_partkey and s_suppkey = l_suppkey
              and l_orderkey = o_orderkey and o_custkey = c_custkey
              and c_nationkey = n1.n_nationkey
              and n1.n_regionkey = r_regionkey and r_name = 'AFRICA'
              and s_nationkey = n2.n_nationkey
              and o_orderdate between date '1995-01-01' a
              and date '1996-12-31' and p_type = 'ECONOMY BRUSHED COPPER'
     ) as all_nations
GROUP BY o_year
ORDER BY o_year
```

9.
```
SELECT nation, o_year, sum(amount) as sum_profit
FROM ( SELECT n_name as nation, extract(year from o_orderdate) as o_year,
              l_extendedprice * (1 - l_discount) -
              ps_supplycost * l_quantity as amount
       FROM   part, supplier, lineitem, partsupp, orders, nation
       WHERE  s_suppkey = l_suppkey and ps_suppkey = l_suppkey
              and ps_partkey = l_partkey and p_partkey = l_partkey
              and o_orderkey = l_orderkey and s_nationkey = n_nationkey
              and p_name like '%blush%'
     ) as profit
GROUP BY nation, o_year
```

```
    ORDER BY nation, o_year desc
```

10.
```
    SELECT  c_custkey, c_name,
            sum(l_extendedprice * (1 - l_discount)) as revenue,
            c_acctbal, n_name, c_address, c_phone, c_comment
    FROM    customer, orders, lineitem, nation
    WHERE   c_custkey = o_custkey and l_orderkey = o_orderkey
            and o_orderdate >= date '19920801'
            and o_orderdate < date '19920801' + interval '3 month'
            and l_returnflag = 'N' and c_nationkey = n_nationkey
    GROUP BY c_custkey, c_name, c_acctbal, c_phone, n_name, c_address,
             c_comment order by revenue desc
```

11.
```
    SELECT ps_partkey, sum(ps_supplycost * ps_availqty) as value
    FROM   partsupp, supplier, nation
    WHERE  ps_suppkey = s_suppkey and s_nationkey = n_nationkey
           and n_name = 'BRAZIL'
    GROUP BY ps_partkey
    HAVING sum(ps_supplycost * ps_availqty) >
                    (
                      SELECT sum(ps_supplycost * ps_availqty) * 2
                      FROM   partsupp, supplier, nation
                      WHERE  ps_suppkey = s_suppkey and s_nationkey = n_nationkey
                             and n_name = 'BRAZIL'
                    )
    ORDER BY value desc
```

12.
```
    SELECT l_shipmode, sum(case when o_orderpriority = '1-URGENT'
                                     or o_orderpriority = '2-HIGH'
                                then 1 else 0 end) as high_line_count,
                       sum(case when o_orderpriority <> '1-URGENT'
                                     and o_orderpriority <> '2-HIGH'
                                then 1 else 0 end) as low_line_count
    FROM   orders, lineitem
    WHERE  o_orderkey = l_orderkey and l_shipmode in ('TRUCK', 'AIR')
           and l_commitdate < l_receiptdate and l_shipdate < l_commitdate
           and l_receiptdate >= date '19960101'
           and l_receiptdate < date '19960101' + interval '1 year'
    GROUP BY l_shipmode
    ORDER BY l_shipmode
```

```
13. SELECT c_count, count(*) as custdist
    FROM ( SELECT c_custkey, count(o_orderkey)
           FROM   customer
                  left outer join orders on c_custkey = o_custkey
                  and o_comment not like '%even%deposits%'
           GROUP BY c_custkey
         ) as c_orders (c_custkey, c_count)
    GROUP BY c_count
    ORDER BY custdist desc, c_count desc


14. SELECT 100.00 * sum(case when p_type like 'PROMO%'
                        then l_extendedprice * (1 - l_discount)
                        else 0 end
                       ) / sum(l_extendedprice * (1 - l_discount))
                    as promo_revenue
    FROM  lineitem, part
    WHEER l_partkey = p_partkey and l_shipdate >= date '19960201'
          and l_shipdate < date '19960201' + interval '1 month'


15. CREATE VIEW revenue (supplier_no, total_revenue) as
           SELECT l_suppkey, sum(l_extendedprice * (1 - l_discount))
           FROM lineitem
           WHERE l_shipdate >= '19960201' and l_shipdate < '19960501'
           GROUP BY l_suppkey


16. SELECT p_brand, p_type, p_size, count(distinct ps_suppkey) as supplier_cnt
    FROM   partsupp, part
    WHERE  p_partkey = ps_partkey and p_brand <> 'Brand#13'
           and p_type not like 'STANDARD%'
           and p_size in (7, 12, 14, 16, 21, 23, 32, 43)
           and ps_suppkey not in ( SELECT s_suppkey
                                   FROM supplier
                                   WHERE s_comment like '%Customer%Complaints%'
                                 )
    GROUP BY p_brand, p_type, p_size
    ORDER BY supplier_cnt desc, p_brand, p_type, p_size
```

```
17. SELECT sum(l_extendedprice) / 7.0 as avg_yearly
    FROM   lineitem, part
    WHERE  p_partkey = l_partkey and p_brand = 'Brand#13'
           and p_container = 'JUMBO PKG'
           and l_quantity < ( SELECT 0.2 * avg(l_quantity)
                              FROM lineitem
                              WHERE l_partkey = p_partkey
                            )


18. SELECT c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice,
           sum(l_quantity)
    FROM   customer, orders, lineitem
    WHERE  o_orderkey in ( SELECT l_orderkey
                           FROM lineitem
                           GROUP BY l_orderkey
                           HAVING sum(l_quantity) > 3
                         )
           and c_custkey = o_custkey
           and o_orderkey = l_orderkey
    GROUP BY c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice
    ORDER BY o_totalprice desc, o_orderdate


19. SELECT sum(l_extendedprice* (1 - l_discount)) as revenue
    FROM   lineitem, part
    WHERE  ( p_partkey = l_partkey and p_brand = 'Brand#13'
             and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
             and l_quantity >= 4 and l_quantity <= 14
             and p_size between 1 and 5
             and l_shipmode in ('AIR', 'AIR REG')
             and l_shipinstruct = 'DELIVER IN PERSON'
           )
           or
           ( p_partkey = l_partkey and p_brand = 'Brand#44'
             and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
             and l_quantity >= 5 and l_quantity <= 15
             and p_size between 1 and 10 and l_shipmode in ('AIR', 'AIR REG')
             and l_shipinstruct = 'DELIVER IN PERSON'
           )
           or
           ( p_partkey = l_partkey and p_brand = 'Brand#53'
             and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
```

```
          and l_quantity >= 6 and l_quantity <= 16
          and p_size between 1 and 15 and l_shipmode in ('AIR', 'AIR REG')
          and l_shipinstruct = 'DELIVER IN PERSON'
        )
```

20.
```
SELECT s_name, s_address
FROM   supplier, nation
WHERE  s_suppkey in ( SELECT distinct (ps_suppkey)
                      FROM partsupp, part
                      WHERE ps_partkey=p_partkey and p_name like 'dim%'
                            and ps_availqty > ( SELECT 0.5 * sum(l_quantity)
                                                FROM lineitem
                                                WHERE l_partkey = ps_partkey
                                                  and l_suppkey = ps_suppkey
                                                  and l_shipdate >= '19970301'
                                                  and l_shipdate <
                                                  date '19970301' +
                                                       interval '1 year'
                                              )
                    )
       and s_nationkey = n_nationkey and n_name = 'ARGENTINA'
ORDER BY s_name
```

21.
```
SELECT s_name, count(*) as numwait
FROM   supplier, lineitem l1, orders, nation
WHERE  s_suppkey = l1.l_suppkey and o_orderkey = l1.l_orderkey
       and o_orderstatus = 'F' and l1.l_receiptdate > l1.l_commitdate
       and exists ( SELECT *
                    FROM lineitem l2
                    WHERE l2.l_orderkey = l1.l_orderkey
                          and l2.l_suppkey <> l1.l_suppkey
                  )
       and not exists ( SELECT *
                        FROM lineitem l3
                        WHERE l3.l_orderkey = l1.l_orderkey
                              and l3.l_suppkey <> l1.l_suppkey
                              and l3.l_receiptdate > l3.l_commitdate
                      )
       and s_nationkey = n_nationkey and n_name = 'BRAZIL'
GROUP BY s_name
ORDER BY numwait desc, s_name
```

```
22. SELECT cntrycode, count(*) as numcust, sum(c_acctbal) as totacctbal
    FROM   ( SELECT substr(c_phone, 1, 2) as cntrycode, c_acctbal
             FROM   customer
             WHERE  substr(c_phone, 1, 2)
                      in ('25', '11', '13', '14', '30', '23', '18')
                    and c_acctbal > ( SELECT avg(c_acctbal)
                                      FROM   customer
                                      WHERE  c_acctbal > 0.00
                                             and substr(c_phone, 1, 2)
                                             in ('25', '11', '13', '14',
                                                 '30', '23','18')
                                    )
                    and not exists ( SELECT *
                                     FROM orders
                                     WHERE o_custkey = c_custkey
                                   )
           ) as vip
    GROUP BY cntrycode
    ORDER BY cntrycode


23. SELECT s_suppkey, s_name, s_address, s_phone, total_revenue
    FROM   supplier, revenue
    WHERE  s_suppkey = supplier_no
           and total_revenue = ( SELECT max(total_revenue) FROM revenue )
    ORDER BY s_suppkey


24. DROP VIEW revenue
```

APÊNDICE C – CARGA SINTÉTICA COM 30 CONSULTAS NA BASE DE DADOS TPC-H

1. Consulta Original:

```
SELECT *
FROM orders
WHERE o_totalprice < ALL (SELECT o_totalprice
                          FROM orders
                          WHERE o_orderpriority = '2-HIGH')
```

Consula Reescrita:

```
SELECT *
FROM orders
WHERE o_totalprice < (SELECT MIN(o_totalprice)
                      FROM orders
                      WHERE o_orderpriority = '2-HIGH')
```

2. Consulta Original:

```
SELECT o_orderkey, o_custkey, o_orderstatus, o_totalprice
FROM orders
WHERE o_totalprice < ANY (SELECT o_totalprice
                          FROM orders
                          WHERE o_orderpriority = '2-HIGH')
```

Consula Reescrita:

```
SELECT o_orderkey, o_custkey, o_orderstatus, o_totalprice
FROM orders
WHERE o_totalprice < (SELECT MAX(o_totalprice)
                      FROM orders
                      WHERE o_orderpriority = '2-HIGH')
```

3. Consulta Original:

```
SELECT o_orderkey, o_custkey, o_orderstatus, o_totalprice
FROM orders
WHERE o_totalprice < SOME (SELECT o_totalprice
                           FROM orders
                           WHERE o_orderpriority = '2-HIGH')
```

Consula Reescrita:

```
SELECT o_orderkey, o_custkey, o_orderstatus, o_totalprice
FROM orders
WHERE o_totalprice < (SELECT MAX(o_totalprice)
                      FROM orders
                      WHERE o_orderpriority = '2-HIGH')
```

4. Consulta Original:

```
SELECT *
FROM part
WHERE p_retailprice > ALL(SELECT p_retailprice
                          FROM part
                          WHERE p_container='JUMBO CAN')
```

Consula Reescrita:

```
SELECT *
FROM part
WHERE p_retailprice > (SELECT MAX(p_retailprice)
                       FROM part
                       WHERE p_container = 'JUMBO CAN')
```

5. Consulta Original:

```
SELECT *
FROM part
WHERE p_retailprice < SOME(SELECT p_retailprice
                           FROM part
                           WHERE p_container='JUMBO CAN')
```

Consula Reescrita:

```
SELECT *
FROM part
WHERE p_retailprice < (SELECT MAX(p_retailprice)
                       FROM part
                       WHERE p_container = 'JUMBO CAN')
```

6. Consulta Original:

```
SELECT *
FROM part
WHERE p_retailprice < ANY(SELECT p_retailprice
                          FROM part
                          WHERE p_container='JUMBO CAN')
```

Consula Reescrita:

```
SELECT *
FROM part
WHERE p_retailprice < (SELECT MAX(p_retailprice)
```

```
                        FROM part
                        WHERE p_container = 'JUMBO CAN')
```

7. Consulta Original:

```
  SELECT distinct(o_orderkey), o_custkey, o_orderstatus, o_totalprice,
         o_orderdate, o_orderpriority, o_clerk, o_shippriority, o_comment
  FROM orders
```

Consula Reescrita:

```
  SELECT o_orderkey, o_custkey, o_orderstatus, o_totalprice,
         o_orderdate, o_orderpriority, o_clerk, o_shippriority, o_comment
  FROM orders
```

8. Consulta Original:

```
  SELECT distinct(c_custkey), c_name, c_address, c_nationkey, c_phone,
         c_acctbal, c_mktsegment, c_comment FROM
  customer
```

Consula Reescrita:

```
  SELECT c_custkey, c_name, c_address, c_nationkey, c_phone,
         c_acctbal, c_mktsegment, c_comment
  FROM customer;
```

9. Consulta Original:

```
   SELECT distinct(c_custkey), c_name, c_address, c_phone,
          c_acctbal, c_mktsegment, c_comment, n_name, r_name
   FROM customer, nation, region
   WHERE c_nationkey=n_nationkey and n_regionkey=r_regionkey
```

Consula Reescrita:

```
  SELECT c_custkey, c_name, c_address, c_phone,
         c_acctbal, c_mktsegment, c_comment, n_name, r_name
  FROM customer, nation, region
  WHERE c_nationkey=n_nationkey and n_regionkey=r_regionkey
```

10. Consulta Original:

```
  SELECT distinct(o_orderkey), o_totalprice, c_custkey, c_name, c_address,
         c_nationkey, c_phone, c_acctbal, c_mktsegment, c_comment
  FROM customer, orders
  WHERE c_custkey=o_custkey
```

Consula Reescrita:

```
  SELECT o_orderkey, o_totalprice, c_custkey, c_name, c_address,
         c_nationkey, c_phone, c_acctbal, c_mktsegment, c_comment
  FROM customer, orders
```

```
      WHERE c_custkey=o_custkey
```

11. Consulta Original:

```
      SELECT distinct(o_orderkey), o_totalprice, c_custkey, c_name
      FROM customer, orders
      WHERE c_custkey= o_custkey
```

Consula Reescrita:

```
      SELECT o_orderkey, o_totalprice, c_custkey, c_name
      FROM customer, orders
      WHERE c_custkey= o_custkey
```

12. Consulta Original:

```
      SELECT max(l_extendedprice)
      FROM lineitem
      GROUP BY l_extendedprice
```

Consula Reescrita:

```
      SELECT max(l_extendedprice)
      FROM lineitem
```

13. Consulta Original:

```
      SELECT o_orderkey, o_custkey, o_orderstatus, o_totalprice, o_orderdate
      FROM orders
      GROUP BY o_orderkey, o_custkey, o_orderstatus, o_totalprice, o_orderdate
```

Consula Reescrita:

```
      SELECT o_orderkey, o_custkey, o_orderstatus, o_totalprice, o_orderdate
      FROM orders
```

14. Consulta Original:

```
      SELECT max(o_totalprice)
      FROM customer, orders WHERE c_custkey=o_custkey
      GROUP BY c_name,o_totalprice
      HAVING o_totalprice <10000
```

Consula Reescrita:

```
      SELECT max(o_totalprice)
      FROM customer, orders WHERE c_custkey=o_custkey
      WHERE c_custkey = o_custkey AND o_totalprice < 10000
```

15. Consulta Original:

```
      SELECT o_orderkey, o_custkey, o_orderstatus, o_totalprice, o_orderdate
      FROM orders
      GROUP BY o_orderkey, o_custkey, o_orderstatus, o_totalprice, o_orderdate
      HAVING o_totalprice <10000
```

Consula Reescrita:

```
    SELECT o_orderkey, o_custkey, o_orderstatus, o_totalprice, o_orderdate
    FROM orders
    WHERE (o_totalprice < 10000)
```

16. Consulta Original:

```
    SELECT *
    FROM part
    WHERE p_mfgr='Manufacturer#5' or p_size=10 or p_retailprice=910
```

Consula Reescrita:

```
    SELECT *
    FROM part
    WHERE p_mfgr = 'Manufacturer#5'
  UNION
    SELECT *
    FROM part
    WHERE p_size = 10
  UNION
    SELECT *
    FROM part
    WHERE p_retailprice > 910
```

17. Consulta Original:

```
    SELECT l_orderkey
    FROM lineitem
    WHERE l_quantity<20 or l_linenumber>5
```

Consula Reescrita:

```
    SELECT l_orderkey
    FROM lineitem
    WHERE l_quantity < 20
  UNION
    SELECT l_orderkey
    FROM lineitem
    WHERE l_linenumber > 5
```

18. Consulta Original:

```
    SELECT *
    FROM lineitem
    WHERE l_quantity*2 = 50
```

Consula Reescrita:

```
    SELECT *
    FROM lineitem
    WHERE l_quantity = (50 / 2)
```

19. Consulta Original:

```
SELECT *
FROM lineitem
WHERE l_quantity*2 = (SELECT max(l_quantity)
                      FROM lineitem)
```

Consula Reescrita:

```
SELECT *
FROM lineitem
WHERE l_quantity = (SELECT max(l_quantity)
                    FROM lineitem) / 2
```

20. Consulta Original:

```
SELECT *
FROM lineitem
WHERE l_quantity + (SELECT min(l_quantity)
                    FROM lineitem) + 10 = (SELECT max(l_quantity)
                                           FROM lineitem)
```

Consula Reescrita:

```
SELECT *
FROM lineitem
WHERE l_quantity = (SELECT max(l_quantity)
                    FROM lineitem) -
                   (SELECT min(l_quantity)
                    FROM lineitem)) - 10
```

21. Consulta Original:

```
SELECT o_orderkey, o_custkey, o_orderstatus, o_totalprice, o_orderdate
FROM orders
WHERE o_totalprice < some (SELECT l_orderkey
                           FROM lineitem
                           WHERE l_quantity +
                                 (SELECT min(l_quantity)
                                  FROM lineitem) + 10 =
                                 (SELECT max(l_quantity)
                                  FROM lineitem) )
```

Consula Reescrita:

```
SELECT o_orderkey, o_custkey, o_orderstatus, o_totalprice, o_orderdate
FROM orders
WHERE o_totalprice < (SELECT MAX(l_orderkey)
                      FROM lineitem
                      WHERE l_quantity = (SELECT max(l_quantity)
                                          FROM lineitem) -
                                         (SELECT min(l_quantity)
```

```
                                        FROM lineitem))) - 10
```

22. Consulta Original:

```
SELECT o_orderkey, o_custkey, o_orderstatus, o_totalprice, o_orderdate
FROM orders
WHERE o_totalprice < any (SELECT l_orderkey
                          FROM lineitem
                          WHERE l_quantity +
                                (SELECT min(l_quantity)
                                 FROM lineitem) + 10 =
                                (SELECT max(l_quantity)
                                 FROM lineitem) )
```

Consula Reescrita:

```
SELECT o_orderkey, o_custkey, o_orderstatus, o_totalprice, o_orderdate
FROM orders
WHERE o_totalprice < (SELECT MAX(l_orderkey)
                      FROM lineitem
                      WHERE l_quantity = (SELECT max(l_quantity)
                                          FROM lineitem) -
                                         (SELECT min(l_quantity)
                                         FROM lineitem))) - 10
```

23. Consulta Original:

```
SELECT o_orderkey, o_custkey, o_orderstatus, o_totalprice, o_orderdate
FROM orders
WHERE o_totalprice > all (SELECT l_orderkey
                          FROM lineitem
                          WHERE l_quantity +
                                (SELECT min(l_quantity)
                                 FROM lineitem) + 10 =
                                (SELECT max(l_quantity)
                                 FROM lineitem) )
```

Consula Reescrita:

```
SELECT o_orderkey, o_custkey, o_orderstatus, o_totalprice, o_orderdate
FROM orders
WHERE o_totalprice > (SELECT MAX(l_orderkey)
                      FROM lineitem
                      WHERE l_quantity = (SELECT max(l_quantity)
                                          FROM lineitem) -
                                         (SELECT min(l_quantity)
                                         FROM lineitem))) - 10
```

24. Consulta Original:

```
SELECT o_totalprice
```

```
    FROM orders
    WHERE o_totalprice > ALL (SELECT o_totalprice
                              FROM orders
                              GROUP BY o_totalprice
                              HAVING o_totalprice > 1000000)
Consula Reescrita:
    SELECT o_totalprice
    FROM orders WHERE o_totalprice > (SELECT MAX(o_totalprice)
                                      FROM orders
                                      WHERE o_totalprice > 1000000)
```

25.
```
Consulta Original:
    SELECT o_totalprice
    FROM orders
    WHERE o_totalprice < SOME (SELECT o_totalprice
                              FROM orders
                              GROUP BY o_totalprice
                              HAVING o_totalprice > 400000)
Consula Reescrita:
    SELECT o_totalprice
    FROM orders WHERE o_totalprice < (SELECT MAX(o_totalprice)
                                      FROM orders
                                      WHERE o_totalprice > 400000)
```

26.
```
Consulta Original:
    SELECT o_totalprice
    FROM orders
    WHERE o_totalprice < any (SELECT o_totalprice
                              FROM orders
                              GROUP BY o_totalprice
                              HAVING o_totalprice > 400000)
Consula Reescrita:
    SELECT o_totalprice
    FROM orders WHERE o_totalprice < (SELECT MAX(o_totalprice)
                                      FROM orders
                                      WHERE o_totalprice > 400000)
```

27.
```
Consulta Original:
    SELECT distinct(o_totalprice)
    FROM orders
    WHERE o_totalprice < any (SELECT o_totalprice
                              FROM orders
                              GROUP BY o_totalprice
```

```
                                HAVING o_totalprice > 400000)
     Consula Reescrita:
       SELECT o_totalprice
       FROM orders WHERE o_totalprice < (SELECT Min(o_totalprice)
                                         FROM orders
                                         WHERE o_totalprice > 400000)
    Consula Reescrita:

28.  Consulta Original:
       SELECT distinct(o_orderkey)
       FROM orders
       WHERE o_totalprice > all (SELECT l_orderkey
                                 FROM lineitem
                                 WHERE l_quantity +
                                       (SELECT min(l_quantity)
                                        FROM lineitem) + 10 =
                                       (SELECT max(l_quantity)
                                        FROM lineitem) )
    Consula Reescrita:
      SELECT o_orderkey
      FROM orders
      WHERE o_totalprice > (SELECT MAX(l_orderkey)
                            FROM lineitem
                            WHERE l_quantity = (SELECT max(l_quantity)
                                                FROM lineitem) -
                                               (SELECT min(l_quantity)
                                               FROM lineitem))) - 10

29. Consulta Original:
      SELECT distinct(o_orderkey)
      FROM orders
      WHERE o_totalprice < SOME (SELECT o_totalprice
                                 FROM orders
                                 WHERE o_orderpriority = '2-HIGH')
    Consula Reescrita:
      SELECT o_orderkey
      FROM orders
      WHERE o_totalprice < (SELECT MAX(o_totalprice)
                            FROM orders
                            WHERE o_orderpriority = '2-HIGH')

30. Consulta Original:
      SELECT distinct(p_partkey)
```

```
  FROM part
  WHERE p_retailprice > ALL(SELECT p_retailprice
                            FROM part
                            WHERE p_container='JUMBO CAN')
Consula Reescrita:
    SELECT p_partkey
    FROM part
    WHERE p_retailprice > (SELECT MAX(p_retailprice)
                           FROM part
                           WHERE p_container = 'JUMBO CAN')
```

APÊNDICE D – CARGA SINTÉTICA COM 30 CONSULTAS NA BASE DE DADOS SIG

1. Consulta Original:

```
SELECT *
FROM graduacao.discente_graduacao
WHERE ira > ALL (SELECT ira
                 FROM graduacao.discente_graduacao g
                 WHERE g.id_matriz_curricular=658149)
```

Consula Reescrita:

```
SELECT *
FROM graduacao.discente_graduacao
WHERE ira > (SELECT MAX(ira)
             FROM graduacao.discente_graduacao AS g
             WHERE g.id_matriz_curricular = 658149)
```

2. Consulta Original:

```
SELECT *
FROM graduacao.discente_graduacao
WHERE ira > SOME (SELECT ira
                  FROM graduacao.discente_graduacao g
                  WHERE g.id_matriz_curricular=658149)
```

Consula Reescrita:

```
SELECT *
FROM graduacao.discente_graduacao
WHERE ira > (SELECT MIN(ira)
             FROM graduacao.discente_graduacao AS g
             WHERE g.id_matriz_curricular = 658149)
```

3. Consulta Original:

```
SELECT *
FROM graduacao.discente_graduacao
WHERE ira < ANY (SELECT ira
                 FROM graduacao.discente_graduacao g
                 WHERE g.id_matriz_curricular=658149)
```

```
Consula Reescrita:
  SELECT *
  FROM graduacao.discente_graduacao
  WHERE ira < (SELECT MAX(ira)
               FROM graduacao.discente_graduacao AS g
               WHERE g.id_matriz_curricular = 658149)
```

4. Consulta Original:

```
  SELECT *
  FROM stricto_sensu.discente_stricto
  WHERE mes_entrada > any (SELECT mes_entrada
                       FROM stricto_sensu.discente_stricto d,
                            stricto_sensu.area_concentracao a
                       WHERE d.id_area_concentracao = a.id_area_concentracao
                             and a.denominacao='EDUCAÇÃO BRASILEIRA')
Consula Reescrita:
  SELECT *
  FROM stricto_sensu.discente_stricto
  WHERE mes_entrada > (SELECT MIN(mes_entrada)
                       FROM stricto_sensu.discente_stricto AS d,
                            stricto_sensu.area_concentracao AS a
                       WHERE d.id_area_concentracao = a.id_area_concentracao
                             AND (a.denominacao = 'EDUCAÇÃO BRASILEIRA')
```

5. Consulta Original:

```
  SELECT *
  FROM ensino.turma
  WHERE capacidade_aluno < any (SELECT capacidade_aluno
                                FROM ensino.turma
                                WHERE codigo = 'A')
Consula Reescrita:
  SELECT *
  FROM ensino.turma
  WHERE capacidade_aluno < (SELECT MAX(capacidade_aluno)
                            FROM ensino.turma
                            WHERE (codigo = 'A'))
```

6. Consulta Original:

```
  SELECT *
  FROM ava.forum
  WHERE data_criacao < ALL(SELECT data_criacao
                           FROM ava.forum
                           WHERE id_usuario=26510)
```

```
Consula Reescrita:
  SELECT *
  FROM ava.forum
  WHERE data_criacao < (SELECT MIN(data_criacao)
                        FROM ava.forum
                        WHERE id_usuario = 26510)
```

7.
```
Consulta Original:
  SELECT *
  FROM ava.forum
  WHERE data_criacao < ANY(SELECT data_criacao
                           FROM ava.forum
                           WHERE id_usuario=26510)
Consula Reescrita:
  SELECT *
  FROM ava.forum
  WHERE data_criacao < (SELECT MAX(data_criacao)
                        FROM ava.forum
                        WHERE id_usuario = 26510)
```

8.
```
Consulta Original:
   SELECT id_discente, ano_ingresso, periodo_ingresso, matricula,
          observacao, id_pessoa, codmerg, id_gestora_academica,
          nivel, status, matricula_temp, id_forma_ingresso,
          ch_integralizada, prazo_conclusao, data_colacao_grau,
          codmergpa, id_curso, id_curriculo, codmergpapos,
          formando, matricula_antiga, tipo, id_foto, id_perfil,
          data_cadastro, codmergcomperve, matricula_substituida,
          id_pessoa_transferencia, id_registro_entrada, codmergppgeec,
          nivel_antigo, id_curso_antigo, periodo_atual,
          id_historico_digital, matricula_sem_digito,
          alterado_cadastrado
   FROM discente
   GROUP BY id_discente, ano_ingresso, periodo_ingresso,
            matricula, observacao, id_pessoa, codmerg,
            id_gestora_academica, nivel, status,
            matricula_temp, id_forma_ingresso,
            ch_integralizada, prazo_conclusao,
            data_colacao_grau, codmergpa, id_curso,
            id_curriculo, codmergpapos, formando,
            matricula_antiga, tipo, id_foto, id_perfil,
            data_cadastro, codmergcomperve,
            matricula_substituida, id_pessoa_transferencia,
```

```
            id_registro_entrada, codmergpppgeec,
            nivel_antigo, id_curso_antigo, periodo_atual,
            id_historico_digital, matricula_sem_digito,
            alterado_cadastrado
```

Consula Reescrita:

```
   SELECT id_discente, ano_ingresso, periodo_ingresso, matricula,
          observacao, id_pessoa, codmerg, id_gestora_academica,
          nivel, status, matricula_temp, id_forma_ingresso,
          ch_integralizada, prazo_conclusao, data_colacao_grau,
          codmergpa, id_curso, id_curriculo, codmergpapos,
          formando, matricula_antiga, tipo, id_foto, id_perfil,
          data_cadastro, codmergcomperve, matricula_substituida,
          id_pessoa_transferencia, id_registro_entrada, codmergpppgeec,
          nivel_antigo, id_curso_antigo, periodo_atual,
          id_historico_digital, matricula_sem_digito,
          alterado_cadastrado
   FROM discente
```

9. Consulta Original:

```
  SELECT distinct(id_discente_graduacao), perfil_inicial, anistiado,
         ira, codmergpa, id_matriz_curricular, pontosvestibular,
         classificacaovestibular, ch_optativa_integralizada,
         ch_nao_atividade_obrig_integ, ch_atividade_obrig_integ,
         ch_aula_integralizada, ch_aula_pendente, ch_lab_pendente,
         ch_estagio_integralizada, ch_estagio_pendente,
         cr_lab_integralizado, cr_lab_pendente, cr_estagio_integralizado,
         cr_estagio_pendente, cr_nao_atividade_obrig_integralizado,
         ch_optativa_pendente, ch_nao_atividade_obrig_pendente,
         ch_atividade_obrig_pendente, cr_nao_atividade_obrig_pendente,
         prazomaximoconclusao, participacao_enad, id_polo,
         escola_conclusao_medio, ano_conclusao_medio,
         cidade_conclusao_medio, uf_conclusao_medio, pais_conclusao_medio,
         ch_lab_integralizada, cr_extra_integralizados,
         ch_total_integralizada, ch_total_pendente,
         cr_total_integralizados, cr_total_pendentes,
         cr_aula_integralizado, cr_aula_pendente,
         ultima_atualizacao_totais, total_atividades_pendentes,
         FROM graduacao.discente_graduacao;
```

Consula Reescrita:

```
   SELECT id_discente_graduacao, perfil_inicial, anistiado,
          ira, codmergpa, id_matriz_curricular, pontosvestibular,
          classificacaovestibular, ch_optativa_integralizada,
          ch_nao_atividade_obrig_integ, ch_atividade_obrig_integ,
```

```
            ch_aula_integralizada, ch_aula_pendente, ch_lab_pendente,
            ch_estagio_integralizada, ch_estagio_pendente,
            cr_lab_integralizado, cr_lab_pendente, cr_estagio_integralizado,
            cr_estagio_pendente, cr_nao_atividade_obrig_integralizado,
            ch_optativa_pendente, ch_nao_atividade_obrig_pendente,
            ch_atividade_obrig_pendente, cr_nao_atividade_obrig_pendente,
            prazomaximoconclusao, participacao_enad, id_polo,
            escola_conclusao_medio, ano_conclusao_medio,
            cidade_conclusao_medio, uf_conclusao_medio, pais_conclusao_medio,
            ch_lab_integralizada, cr_extra_integralizados,
            ch_total_integralizada, ch_total_pendente,
            cr_total_integralizados, cr_total_pendentes,
            cr_aula_integralizado, cr_aula_pendente,
            ultima_atualizacao_totais, total_atividades_pendentes,
            FROM graduacao.discente_graduacao;

10. Consulta Original:
      SELECT distinct(id_turma), ano, periodo, id_situacao_turma,
             capacidade_aluno, total_matriculados, "@hora_livre", local,
             data_inicio, data_fim, id_disciplina, codmerg, codigo,
             id_plano_curso, codmergpa, total_reservados, total_espera,
             status, pro_basica, vagas_reservadas, descricao_horario,
             repeticaohorario, codmergpapos, id_convenio,
             id_especializacao_turma_entrada, id_turma_bloco, observacao,
             distancia, id_polo, id_curso, tipo, processada, total_solicitacoes,
             processamento_rematricula, codmergppgeec, id_registro_cadastro,
             data_cadastro, id_registro_atualizacao, data_alteracao,
             id_campus_ies, processada_rematricula, id_usuario_consolidacao,
             data_consolidacao, "@nomes_docentes", x, id_turma_agrupadora,
             agrupadora, permite_consolidacao_docente, horario_extenso,
             unidade_academica, sub_unidade_academica
             FROM ensino.turma;
    Consula Reescrita:
      SELECT id_turma, ano, periodo, id_situacao_turma,
             capacidade_aluno, total_matriculados, "@hora_livre", local,
             data_inicio, data_fim, id_disciplina, codmerg, codigo,
             id_plano_curso, codmergpa, total_reservados, total_espera,
             status, pro_basica, vagas_reservadas, descricao_horario,
             repeticaohorario, codmergpapos, id_convenio,
             id_especializacao_turma_entrada, id_turma_bloco, observacao,
             distancia, id_polo, id_curso, tipo, processada, total_solicitacoes,
             processamento_rematricula, codmergppgeec, id_registro_cadastro,
             data_cadastro, id_registro_atualizacao, data_alteracao,
```

```
            id_campus_ies, processada_rematricula, id_usuario_consolidacao,
            data_consolidacao, "@nomes_docentes", x, id_turma_agrupadora,
            agrupadora, permite_consolidacao_docente, horario_extenso,
            unidade_academica, sub_unidade_academica
            FROM ensino.turma;
```

11. Consulta Original:

```
  SELECT *
  FROM graduacao.solicitacao_matricula
  WHERE ano + 1= 2010
```

Consula Reescrita:

```
  SELECT *
  FROM graduacao.solicitacao_matricula
  WHERE ano + 1= 2010;
```

12. Consulta Original:

```
  SELECT *
  FROM graduacao.solicitacao_matricula
  WHERE ano +2 = DATE_PART('YEAR', CURRENT_TIMESTAMP)
```

Consula Reescrita:

```
  SELECT *
  FROM graduacao.solicitacao_matricula
  WHERE ano = DATE_PART('YEAR',CURRENT_TIMESTAMP) { 2
```

13. Consulta Original:

```
  SELECT *
  FROM graduacao.discente_graduacao
  WHERE ira - 1 = (SELECT avg(ira)
                   FROM graduacao.discente_graduacao)
```

Consula Reescrita:

```
  SELECT *
  FROM graduacao.discente_graduacao
  WHERE ira = (SELECT avg(ira)
               FROM graduacao.discente_graduacao) + 1
```

14. Consulta Original:

```
  SELECT *
  FROM graduacao.discente_graduacao
  WHERE ira * 2 = (SELECT avg(ira)
                   FROM graduacao.discente_graduacao)
```

Consula Reescrita:

```
  SELECT *
  FROM graduacao.discente_graduacao
```

```
    WHERE ira = (SELECT avg(ira)
                 FROM graduacao.discente_graduacao) / 2
```

15. Consulta Original:

```
    SELECT *
    FROM ensino.turma t, ensino.docente_turma dt, rh.servidor s
    WHERE capacidade_aluno*2 = total_matriculados and
          s.id_servidor=dt.id_docente and t.id_turma=dt.id_turma
```

Consula Reescrita:

```
    SELECT *
    FROM ensino.turma AS t, ensino.docente_turma AS dt, rh.servidor AS s
    WHERE capacidade_aluno = total_matriculados / 2 AND
          s.id_servidor = dt.id_docente AND t.id_turma = dt.id_turma
```

16. Consulta Original:

```
    SELECT *
    FROM stricto_sensu.discente_stricto
    WHERE (mes_entrada+12)/2 = 6
```

Consula Reescrita:

```
    SELECT *
    FROM stricto_sensu.discente_stricto
    WHERE mes_entrada = (6 * 2) - 12
```

17. Consulta Original:

```
    SELECT *
    FROM stricto_sensu.discente_stricto
    WHERE mes_entrada + id_origem_discente = 6
```

Consula Reescrita:

```
    SELECT *
    FROM stricto_sensu.discente_stricto
    WHERE mes_entrada = 6 - id_origem_discente
```

18. Consulta Original:

```
    SELECT t.id_turma
    FROM ensino.turma t, ensino.docente_turma dt, rh.servidor s
    WHERE capacidade_aluno*2 = 10 and
          s.id_servidor=dt.id_docente and
          t.id_turma=dt.id_turma
```

Consula Reescrita:

```
    SELECT t.id_turma
    FROM ensino.turma AS t, ensino.docente_turma AS dt, rh.servidor AS s
    WHERE capacidade_aluno = (10 / 2) AND
          s.id_servidor = dt.id_docente AND
```

```
        t.id_turma = dt.id_turma
```

19. Consulta Original:

```
  SELECT *
  FROM discente
  WHERE ano_ingresso=2010 or nivel='D' or status=6
```

Consula Reescrita:

```
    SELECT *
    FROM discente
    WHERE ano_ingresso = 2010
  UNION
    SELECT *
    FROM discente
    WHERE nivel = 'D'
  UNION
    SELECT *
    FROM discente
    WHERE status = 6
```

20. Consulta Original:

```
  SELECT id_servidor, siape, id_pessoa, id_atividade, digito_siape,
         id_escolaridade, id_ativo, id_situacao, id_categoria,
         id_cargo, uorg, lotacao, ultima_atualizacao, auxilio_transporte,
         uorgsiapecad, id_unidade, id_formacao, regime_trabalho,
         dedicacao_exclusiva, tipo_vinculo, id_foto, id_perfil,
         id_pessoa_transferencia, data_desligamento, id_classe_funcional,
         admissao, referencia_nivel_padrao, id_unidade_lotacao,
         matricula_interna, nome_identificacao
  FROM ensino.docente_turma dt, rh.servidor s
  WHERE s.id_servidor=dt.id_docente
```

Consula Reescrita:

```
  SELECT id_servidor, siape, id_pessoa, id_atividade, digito_siape,
         id_escolaridade, id_ativo, id_situacao, id_categoria,
         id_cargo, uorg, lotacao, ultima_atualizacao, auxilio_transporte,
         uorgsiapecad, id_unidade, id_formacao, regime_trabalho,
         dedicacao_exclusiva, tipo_vinculo, id_foto, id_perfil,
         id_pessoa_transferencia, data_desligamento, id_classe_funcional,
         admissao, referencia_nivel_padrao, id_unidade_lotacao,
         matricula_interna, nome_identificacao
  FROM ensino.docente_turma dt, rh.servidor s
  WHERE s.id_servidor=dt.id_docente
```

21. Consulta Original:

```
  SELECT *
  FROM graduacao.discente_graduacao
  WHERE ira > ALL (SELECT ira
                   FROM graduacao.discente_graduacao g
                   WHERE g.id_matriz_curricular=658149)
        and (ira - 1 = (SELECT avg(ira)
                        FROM graduacao.discente_graduacao))
Consula Reescrita:
  SELECT *
  FROM graduacao.discente_graduacao
  WHERE ira > (SELECT MAX(ira)
               FROM graduacao.discente_graduacao AS g
               WHERE g.id_matriz_curricular = 658149)
        AND (ira = (SELECT avg(ira) FROM graduacao.discente_graduacao) + 1)
```

22. Consulta Original:

```
  SELECT *
  FROM graduacao.discente_graduacao
  WHERE ira > SOME (SELECT ira
                   FROM graduacao.discente_graduacao g
                   WHERE g.id_matriz_curricular=658149)
        and (ira - 1 = (SELECT avg(ira)
                        FROM graduacao.discente_graduacao))
Consula Reescrita:
  SELECT *
  FROM graduacao.discente_graduacao
  WHERE ira > (SELECT MIN(ira)
               FROM graduacao.discente_graduacao AS g
               WHERE g.id_matriz_curricular = 658149)
        AND (ira = (SELECT avg(ira) FROM graduacao.discente_graduacao) + 1)
```

23. Consulta Original:

```
  SELECT distinct(id_discente_graduacao), perfil_inicial, anistiado,
        ira, codmergpa, id_matriz_curricular, pontosvestibular,
        classificacaovestibular, ch_optativa_integralizada,
        ch_nao_atividade_obrig_integ, ch_atividade_obrig_integ,
        ch_aula_integralizada, ch_aula_pendente, ch_lab_pendente,
        ch_estagio_integralizada, ch_estagio_pendente,
        cr_lab_integralizado, cr_lab_pendente, cr_estagio_integralizado,
        cr_estagio_pendente, cr_nao_atividade_obrig_integralizado,
        ch_optativa_pendente, ch_nao_atividade_obrig_pendente,
        ch_atividade_obrig_pendente, cr_nao_atividade_obrig_pendente,
        prazomaximoconclusao, participacao_enad, id_polo,
```

```
        escola_conclusao_medio, ano_conclusao_medio,
        cidade_conclusao_medio, uf_conclusao_medio, pais_conclusao_medio,
        ch_lab_integralizada, cr_extra_integralizados,
        ch_total_integralizada, ch_total_pendente,
        cr_total_integralizados, cr_total_pendentes,
        cr_aula_integralizado, cr_aula_pendente,
        ultima_atualizacao_totais, total_atividades_pendentes,
        FROM graduacao.discente_graduacao
        WHERE ira - 1 > (SELECT avg(ira)
                         FROM graduacao.discente_graduacao)
Consula Reescrita:
  SELECT id_discente_graduacao, perfil_inicial, anistiado,
         ira, codmergpa, id_matriz_curricular, pontosvestibular,
         classificacaovestibular, ch_optativa_integralizada,
         ch_nao_atividade_obrig_integ, ch_atividade_obrig_integ,
         ch_aula_integralizada, ch_aula_pendente, ch_lab_pendente,
         ch_estagio_integralizada, ch_estagio_pendente,
         cr_lab_integralizado, cr_lab_pendente, cr_estagio_integralizado,
         cr_estagio_pendente, cr_nao_atividade_obrig_integralizado,
         ch_optativa_pendente, ch_nao_atividade_obrig_pendente,
         ch_atividade_obrig_pendente, cr_nao_atividade_obrig_pendente,
         prazomaximoconclusao, participacao_enad, id_polo,
         escola_conclusao_medio, ano_conclusao_medio,
         cidade_conclusao_medio, uf_conclusao_medio, pais_conclusao_medio,
         ch_lab_integralizada, cr_extra_integralizados,
         ch_total_integralizada, ch_total_pendente,
         cr_total_integralizados, cr_total_pendentes,
         cr_aula_integralizado, cr_aula_pendente,
         ultima_atualizacao_totais, total_atividades_pendentes,
  FROM graduacao.discente_graduacao
  WHERE ira > (SELECT avg(ira)
               FROM graduacao.discente_graduacao) + 1
```

24. Consulta Original:

```
  SELECT distinct(id_discente_graduacao), perfil_inicial, anistiado,
         ira, codmergpa, id_matriz_curricular, pontosvestibular,
         classificacaovestibular, ch_optativa_integralizada,
         ch_nao_atividade_obrig_integ, ch_atividade_obrig_integ,
         ch_aula_integralizada, ch_aula_pendente, ch_lab_pendente,
         ch_estagio_integralizada, ch_estagio_pendente,
         cr_lab_integralizado, cr_lab_pendente, cr_estagio_integralizado,
         cr_estagio_pendente, cr_nao_atividade_obrig_integralizado,
         ch_optativa_pendente, ch_nao_atividade_obrig_pendente,
```

```
            ch_atividade_obrig_pendente, cr_nao_atividade_obrig_pendente,
            prazomaximoconclusao, participacao_enad, id_polo,
            escola_conclusao_medio, ano_conclusao_medio,
            cidade_conclusao_medio, uf_conclusao_medio, pais_conclusao_medio,
            ch_lab_integralizada, cr_extra_integralizados,
            ch_total_integralizada, ch_total_pendente,
            cr_total_integralizados, cr_total_pendentes,
            cr_aula_integralizado, cr_aula_pendente,
            ultima_atualizacao_totais, total_atividades_pendentes,
      FROM graduacao.discente_graduacao
      WHERE ira > ALL (SELECT ira
                       FROM graduacao.discente_graduacao g
                       WHERE g.id_matriz_curricular=658149)
   Consula Reescrita:
      SELECT id_discente_graduacao, perfil_inicial, anistiado,
            ira, codmergpa, id_matriz_curricular, pontosvestibular,
            classificacaovestibular, ch_optativa_integralizada,
            ch_nao_atividade_obrig_integ, ch_atividade_obrig_integ,
            ch_aula_integralizada, ch_aula_pendente, ch_lab_pendente,
            ch_estagio_integralizada, ch_estagio_pendente,
            cr_lab_integralizado, cr_lab_pendente, cr_estagio_integralizado,
            cr_estagio_pendente, cr_nao_atividade_obrig_integralizado,
            ch_optativa_pendente, ch_nao_atividade_obrig_pendente,
            ch_atividade_obrig_pendente, cr_nao_atividade_obrig_pendente,
            prazomaximoconclusao, participacao_enad, id_polo,
            escola_conclusao_medio, ano_conclusao_medio,
            cidade_conclusao_medio, uf_conclusao_medio, pais_conclusao_medio,
            ch_lab_integralizada, cr_extra_integralizados,
            ch_total_integralizada, ch_total_pendente,
            cr_total_integralizados, cr_total_pendentes,
            cr_aula_integralizado, cr_aula_pendente,
            ultima_atualizacao_totais, total_atividades_pendentes,
      FROM graduacao.discente_graduacao
      WHERE ira > (SELECT max(ira)
                   FROM graduacao.discente_graduacao g
                   WHERE g.id_matriz_curricular=658149)

25.  Consulta Original:
      SELECT distinct(id_discente_graduacao), perfil_inicial, anistiado,
            ira, codmergpa, id_matriz_curricular, pontosvestibular,
            classificacaovestibular, ch_optativa_integralizada,
            ch_nao_atividade_obrig_integ, ch_atividade_obrig_integ,
            ch_aula_integralizada, ch_aula_pendente, ch_lab_pendente,
```

```
        ch_estagio_integralizada, ch_estagio_pendente,
        cr_lab_integralizado, cr_lab_pendente, cr_estagio_integralizado,
        cr_estagio_pendente, cr_nao_atividade_obrig_integralizado,
        ch_optativa_pendente, ch_nao_atividade_obrig_pendente,
        ch_atividade_obrig_pendente, cr_nao_atividade_obrig_pendente,
        prazomaximoconclusao, participacao_enad, id_polo,
        escola_conclusao_medio, ano_conclusao_medio,
        cidade_conclusao_medio, uf_conclusao_medio, pais_conclusao_medio,
        ch_lab_integralizada, cr_extra_integralizados,
        ch_total_integralizada, ch_total_pendente,
        cr_total_integralizados, cr_total_pendentes,
        cr_aula_integralizado, cr_aula_pendente,
        ultima_atualizacao_totais, total_atividades_pendentes,
  FROM graduacao.discente_graduacao
  WHERE ira > ALL (SELECT ira
                   FROM graduacao.discente_graduacao g
                   WHERE g.id_matriz_curricular=658149) and
              ira - 1 > (SELECT avg(ira)
                         FROM graduacao.discente_graduacao)
Consula Reescrita:
  SELECT id_discente_graduacao, perfil_inicial, anistiado,
        ira, codmergpa, id_matriz_curricular, pontosvestibular,
        classificacaovestibular, ch_optativa_integralizada,
        ch_nao_atividade_obrig_integ, ch_atividade_obrig_integ,
        ch_aula_integralizada, ch_aula_pendente, ch_lab_pendente,
        ch_estagio_integralizada, ch_estagio_pendente,
        cr_lab_integralizado, cr_lab_pendente, cr_estagio_integralizado,
        cr_estagio_pendente, cr_nao_atividade_obrig_integralizado,
        ch_optativa_pendente, ch_nao_atividade_obrig_pendente,
        ch_atividade_obrig_pendente, cr_nao_atividade_obrig_pendente,
        prazomaximoconclusao, participacao_enad, id_polo,
        escola_conclusao_medio, ano_conclusao_medio,
        cidade_conclusao_medio, uf_conclusao_medio, pais_conclusao_medio,
        ch_lab_integralizada, cr_extra_integralizados,
        ch_total_integralizada, ch_total_pendente,
        cr_total_integralizados, cr_total_pendentes,
        cr_aula_integralizado, cr_aula_pendente,
        ultima_atualizacao_totais, total_atividades_pendentes,
  FROM graduacao.discente_graduacao
  WHERE ira > (SELECT max(ira)
               FROM graduacao.discente_graduacao g
               WHERE g.id_matriz_curricular=658149) and
        ira > (SELECT avg(ira)
```

```
                FROM graduacao.discente_graduacao) + 1
```

26. Consulta Original:

```
  SELECT t.id_turma, t.ano, t.periodo, t.id_situacao_turma,
         t.capacidade_aluno, t.total_matriculados, t."@hora_livre",
         t.local, t.data_inicio, t.data_fim, t.id_disciplina,
         t.codmerg, t.codigo, t.id_plano_curso, t.codmergpa,
         t.total_reservados, t.total_espera, t.status, t.pro_basica,
         t.vagas_reservadas, t.descricao_horario, t.repeticaohorario,
         t.codmergpapos, t.id_convenio, t.id_especializacao_turma_entrada,
         t.id_turma_bloco, t.observacao, t.distancia, t.id_polo,
         t.id_curso, t.tipo, t.processada, t.total_solicitacoes,
         t.processamento_rematricula, t.codmergpppgeec, t.id_registro_cadastro,
         t.data_cadastro, t.id_registro_atualizacao, t.data_alteracao,
         t.id_campus_ies, t.processada_rematricula,
         t.id_usuario_consolidacao, t.data_consolidacao,
         t."@nomes_docentes", t.x, t.id_turma_agrupadora, t.agrupadora,
         t.permite_consolidacao_docente, t.horario_extenso,
         t.unidade_academica, t.sub_unidade_academica
         FROM ensino.turma t, ensino.docente_turma dt, rh.servidor s
         WHERE capacidade_aluno*2 = total_matriculados and
               s.id_servidor=dt.id_docente and t.id_turma=dt.id_turma
```

Consula Reescrita:

```
  SELECT t.id_turma, t.ano, t.periodo, t.id_situacao_turma,
         t.capacidade_aluno, t.total_matriculados, t."@hora_livre",
         t.local, t.data_inicio, t.data_fim, t.id_disciplina,
         t.codmerg, t.codigo, t.id_plano_curso, t.codmergpa,
         t.total_reservados, t.total_espera, t.status, t.pro_basica,
         t.vagas_reservadas, t.descricao_horario, t.repeticaohorario,
         t.codmergpapos, t.id_convenio, t.id_especializacao_turma_entrada,
         t.id_turma_bloco, t.observacao, t.distancia, t.id_polo,
         t.id_curso, t.tipo, t.processada, t.total_solicitacoes,
         t.processamento_rematricula, t.codmergpppgeec, t.id_registro_cadastro,
         t.data_cadastro, t.id_registro_atualizacao, t.data_alteracao,
         t.id_campus_ies, t.processada_rematricula,
         t.id_usuario_consolidacao, t.data_consolidacao,
         t."@nomes_docentes", t.x, t.id_turma_agrupadora, t.agrupadora,
         t.permite_consolidacao_docente, t.horario_extenso,
         t.unidade_academica, t.sub_unidade_academica
         FROM ensino.turma t, ensino.docente_turma dt, rh.servidor s
         WHERE capacidade_aluno = total_matriculados/2 and
               s.id_servidor=dt.id_docente and t.id_turma=dt.id_turma
```

27. Consulta Original:

```
SELECT id_discente, ano_ingresso, periodo_ingresso, matricula,
       observacao, id_pessoa, codmerg, id_gestora_academica, nivel,
       status, matricula_temp, id_forma_ingresso, ch_integralizada,
       prazo_conclusao, data_colacao_grau, codmergpa, id_curso,
       id_curriculo, codmergpapos, formando, matricula_antiga, tipo,
       id_foto, id_perfil, data_cadastro, codmergcomperve,
       matricula_substituida, id_pessoa_transferencia,
       id_registro_entrada, codmergppgeec, nivel_antigo,
       id_curso_antigo, periodo_atual, id_historico_digital,
       matricula_sem_digito, alterado_cadastrado
       FROM discente
       GROUP BY id_discente, ano_ingresso, periodo_ingresso,
       matricula, observacao, id_pessoa, codmerg,
       id_gestora_academica, nivel, status, matricula_temp,
       id_forma_ingresso, ch_integralizada, prazo_conclusao,
       data_colacao_grau, codmergpa, id_curso, id_curriculo,
       codmergpapos, formando, matricula_antiga, tipo, id_foto,
       id_perfil, data_cadastro, codmergcomperve, matricula_substituida,
       id_pessoa_transferencia, id_registro_entrada, codmergppgeec,
       nivel_antigo, id_curso_antigo, periodo_atual,
       id_historico_digital, matricula_sem_digito, alterado_cadastrado
       HAVING ano_ingresso = 2011
```

Consula Reescrita:

```
SELECT id_discente, ano_ingresso, periodo_ingresso, matricula,
       observacao, id_pessoa, codmerg, id_gestora_academica, nivel,
       status, matricula_temp, id_forma_ingresso, ch_integralizada,
       prazo_conclusao, data_colacao_grau, codmergpa, id_curso,
       id_curriculo, codmergpapos, formando, matricula_antiga, tipo,
       id_foto, id_perfil, data_cadastro, codmergcomperve,
       matricula_substituida, id_pessoa_transferencia,
       id_registro_entrada, codmergppgeec, nivel_antigo,
       id_curso_antigo, periodo_atual, id_historico_digital,
       matricula_sem_digito, alterado_cadastrado
       FROM discente
       WHERE ano_ingresso = 2011
```

28. Consulta Original:

```
SELECT DISTINCT(id_discente), ano_ingresso, periodo_ingresso, matricula,
       observacao, id_pessoa, codmerg, id_gestora_academica, nivel,
       status, matricula_temp, id_forma_ingresso, ch_integralizada,
       prazo_conclusao, data_colacao_grau, codmergpa, id_curso,
       id_curriculo, codmergpapos, formando, matricula_antiga, tipo,
```

```
        id_foto, id_perfil, data_cadastro, codmergcomperve,
        matricula_substituida, id_pessoa_transferencia, id_registro_entrada,
        codmergpppgeec, nivel_antigo, id_curso_antigo, periodo_atual,
        id_historico_digital, matricula_sem_digito, alterado_cadastrado
        FROM discente
  GROUP BY id_discente, ano_ingresso, periodo_ingresso, matricula,
           observacao, id_pessoa, codmerg, id_gestora_academica,
           nivel, status, matricula_temp, id_forma_ingresso,
           ch_integralizada, prazo_conclusao, data_colacao_grau,
           codmergpa, id_curso, id_curriculo, codmergpapos, formando,
           matricula_antiga, tipo, id_foto, id_perfil, data_cadastro,
           codmergcomperve, matricula_substituida, id_pessoa_transferencia,
           id_registro_entrada, codmergpppgeec, nivel_antigo, id_curso_antigo,
           periodo_atual, id_historico_digital, matricula_sem_digito,
           alterado_cadastrado
  HAVING ano_ingresso = 2011
Consula Reescrita:
  SELECT id_discente, ano_ingresso, periodo_ingresso, matricula,
         observacao, id_pessoa, codmerg, id_gestora_academica, nivel,
         status, matricula_temp, id_forma_ingresso, ch_integralizada,
         prazo_conclusao, data_colacao_grau, codmergpa, id_curso,
         id_curriculo, codmergpapos, formando, matricula_antiga, tipo,
         id_foto, id_perfil, data_cadastro, codmergcomperve,
         matricula_substituida, id_pessoa_transferencia, id_registro_entrada,
         codmergpppgeec, nivel_antigo, id_curso_antigo, periodo_atual,
         id_historico_digital, matricula_sem_digito, alterado_cadastrado
  FROM discente
  WHERE ano_ingresso = 2011

29. Consulta Original:
  SELECT *
  FROM comum.pessoa
  WHERE UPPER(pais_nacionalidade) = 'BRASILEIRA'
Consula Reescrita:
    SELECT *
    FROM comum.pessoa
    WHERE pais_nacionalidade = 'brasileira'
  UNION
    SELECT *
    FROM comum.pessoa
    WHERE (pais_nacionalidade = 'BRASILEIRA'

30. Consulta Original:
```

```
   SELECT *
   FROM comum.pessoa
   WHERE id_estado_civil = 7 OR pais_nacionalidade = 'brasileira'
Consula Reescrita:
     SELECT *
     FROM comum.pessoa
     WHERE id_estado_civil = 7
   UNION
     SELECT *
     FROM comum.pessoa
     WHERE pais_nacionalidade = 'brasileira'
```