Pontifícia Universidade Católica
do Rio de Janeiro

Rafael Pereira de Oliveira

Combinação e seleção automática de ações de sintonia fina

Tese de Doutorado

Tese apresentada como requisito parcial para obtenção do grau de Doutor pelo Programa de Pós–graduação em Informática da PUC-Rio.

Orientador: Prof. Sérgio Lifschitz

Rio de Janeiro
Setembro de 2019

Pontifícia Universidade Católica
do Rio de Janeiro

Rafael Pereira de Oliveira

Combinação e seleção automática de ações de sintonia fina

Tese apresentada como requisito parcial para obtenção do grau de Doutor pelo Programa de Pós–graduação em Informática da PUC-Rio. Aprovada pela Comissão Examinadora abaixo.

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

Prof. Carlos José Pereira de Lucena
Departamento de Informática – PUC-Rio

Prof. Marcos Kalinowski
Departamento de Informática – PUC-Rio

Prof.ª Fernanda Araujo Baião Amorim
Departamento de Engenharia Industrial – PUC-Rio

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

Prof.ª Ana Carolina Brito de Almeida
Universidade Estadual do Rio de Janeiro – UERJ

Rio de Janeiro, 09 de Setembro de 2019

Rafael Pereira de Oliveira

O autor recebeu seu grau de Mestre em Informática pelo Departamento de Informática (INF) da Pontifícia Universidade Católica do Rio de Janeiro em 2015. Durante o seu curso de doutorado recebeu bolsa CNPQ, foi selecionado para o programa de intercâmbio CAPES PDSE e estudou por onze meses entre 01/09/2017e 30/07/2018 no New Jersey Institute of Tecnology (NJIT), New Jersey, Estados Unidos. Ele também foi selecionado pelo programa NII International Internship Program para estudar por quatro meses entre 01/03/2019 e 30/06/2019 no National Institute of Informatics (NII), Tóquio, Japão.

Ficha Catalográfica

Oliveira, Rafael Pereira de

Combinação e seleção automática de ações de sintonia fina / Rafael Pereira de Oliveira; orientador: Sérgio Lifschitz. – Rio de janeiro: PUC-Rio, Departamento de Informática, 2019.

v., 127 f: il. color. ; 30 cm

Tese (doutorado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui bibliografia

1. Informática – Teses. 2. Banco de dados;. 3. Sintonia fina;. 4. Sintonia fina global;. 5. Método de sintonia fina;. 6. Combinação de ações;. 7. Geração de ações;. 8. Seleção de ações;. I. Lifschitz, Sérgio. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

I must not fear. Fear is the mind-killer. Fear is the little-death that brings total obliteration. I will face my fear. I will permit it to pass over me and through me. And when it has gone past I will turn the inner eye to see its path. Where the fear has gone there will be nothing. Only I will remain.

Frank Herbert, *Dune*

Agradecimentos

Verba volant, scripta manent.

Agradeço à minha esposa Rayanny, que sempre me deu apoio durante esta longa caminhada, e enfrentou todos os desafios ao meu lado.

Agradeço ao meu orientador, Professor Sérgio Lifschitz por, pacientemente, me guiar durante todos estes anos. Aos Professores Daniel Schwabe e Lucena que contribuíram e acompanharam este trabalho desde o início. Aos professores Javam Machado, Marcos Kalinowski, Fernanda Baião, e Ana Carolina por participarem da comissão examinadora.

Agradeço aos professores Vincent Oria e Michael Houle por me receberem em suas respectivas universidades durante meus intercâmbios. Foram momentos muito importantes para a minha formação, assim como seus ensinamentos.

Agradeço a todos os meus colegas do laboratório BioBD pelo apoio, ajuda e discussões, em especial a Ema e Mariana.

Agradeço a todos os meus amigos e familiares que, diretamente ou indiretamente, contribuíram nesta caminhada. Em especial a minha amiga e professora Alison, que por muitas vezes me fez rir dos problemas e dificuldades.

Agradeço ao meu pai Nilton, e ao meu irmão Tiago por todo o incentivo e ajuda desde sempre. Meu eterno agradecimento à minha Mãe que foi (literalmente) a minha primeira e melhor professora. Ela me ensinou quatro coisas: ler, escrever, contar, e respeitar. Tudo que aprendi depois que sai da escola Nossa Senhora de Lourdes foi consequência do que aprendi dentro dela.

Last but not least, agradeço aos amigos do grupo *Sexta-Feira Santa*, exemplos de dedicação, foco e trabalho árduo. Entre eles, meu grande amigo Bruno Olivieri, que sempre me apoiou e esteve presente, até quando estando longe. Muito obrigado por todas as risadas, conselhos sinceros, e manhãs de domingo na APA. Como nos ensinou Bruno certa vez: *"O bonde passa, sobe quem tem o ticket"*.

O presente trabalho foi realizado com o apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

Agradeço também à PUC-Rio, CNPQ, New Jersey Institute of Informatics, e National Institute of Informatics pelo apoio financeiro.

Resumo

Oliveira, Rafael Pereira de; Lifschitz, Sérgio. **Combinação e seleção automática de ações de sintonia fina**. Rio de Janeiro, 2019. 127p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

O processo de combinação de ações de sintonia fina não possui nem uma formulação precisa, nem uma abordagem formal para solucioná-lo. É necessário definir o que combinar dentre as múltiplas ações existentes e, uma vez escolhidas, como compor de maneira que as restrições sejam verificadas. Trata-se de um problema complexo e relevante na área de bancos de dados, tanto para soluções manuais pelo DBA como automáticas, por meio de softwares especializados. Isto ocorre pois os diferentes tipos de ações de sintonia possuem estratégias distintas para alcançar o objetivo em comum. Esta tese propõe um método automático para geração e seleção de soluções combinadas de sintonia fina para bancos de dados relacionais. Discute-se como combinar soluções e respeitar as restrições tecnológicas e recursos computacionais disponíveis. Por fim, apresenta-se uma implementação e avaliação utilizando três SGBDs de mercado relevantes, em que mostramos tanto a eficácia como a eficiência do método proposto. Os resultados mostraram que o método é capaz de produzir soluções combinadas válidas mais eficientes que soluções locais independentes.

Palavras-chave

Banco de dados; Sintonia fina; Sintonia fina global; Método de sintonia fina; Combinação de ações; Geração de ações; Seleção de ações;

Abstract

Oliveira, Rafael Pereira de; Lifschitz, Sérgio (Advisor). **Automatic combination and selection of database tuning actions**. Rio de Janeiro, 2019. 127p. Tese de doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

The process of combining database tuning actions has neither a precise formulation nor a formal approach to solving it. It is necessary to define what to combine among multiple existing operations and, once chosen, how to compose so that constraints can be verified. It is a complex and relevant problem in the database research area, both for the DBA manual solutions, and automatic ones using specialized software. It is important because the different types of tuning actions have different strategies to achieve a common goal. This thesis proposes an automated method for generating and selecting combined tuning solutions for relational databases. It discusses how to mix solutions and still respect both the technological constraints and available computational resources. Finally, we present an implementation and evaluation using three relevant market DBMSs, where we show both the effectiveness and the efficiency of the proposed method. The results showed that the technique is capable of producing combined solutions that are more efficient than independent local solutions.

Keywords

Database; Tuning; global database tuning; Database Tuning Method; Combination Method; Tuning action generation; Tuning action selection;

Sumário

1 Introdução **15**
1.1 Hipótese 16
1.2 O problema de pesquisa 16
1.3 Objetivo Geral 16
1.3.1 Questões de pesquisa 16
1.3.2 Objetivos Específicos 17
1.4 Avaliação 17
1.5 Contribuições 17
1.6 Organização da Tese 18

2 Conceitos Fundamentais **19**
2.1 Modelagem do banco de dados e a tarefa de sintonia fina 19
2.2 Técnicas de sintonia fina 21
2.2.1 Índices 21
2.2.2 Índices completos 22
2.2.3 Índices parciais 22
2.2.4 Visões materializadas 23
2.3 Branch-and-Bound 24
2.4 Sistema multiagentes 25
2.5 Resumo do Capítulo 26

3 Trabalhos relacionados e revisão da literatura **27**
3.1 Principais Trabalhos Relacionados 27
3.2 Algoritmos para sintonia fina local 29
3.3 Técnicas de aprendizado de máquina 31
3.4 Ferramentas de sintonia fina automática proprietárias 32
3.4.1 Automatic SQL Tuning (Oracle) 32
3.4.2 Oracle Autonomous Database 33
3.4.3 SQL Server Automatic Tuning 34
3.5 Ferramentas de sintonia fina local com código aberto 34
3.5.1 Dexter 34
3.5.2 PoWA 35

4 Método para combinação e seleção automática de ações de sintonia fina **36**
4.1 Definições 36
4.1.1 Solução combinada 36
4.1.2 Solução inexpressiva 37
4.1.3 Soluções mutuamente exclusivas 37
4.1.4 Solução inválida 38
4.1.5 Diferentes recursos usados como limites pelo método 39
4.2 O método de combinação e seleção 39
4.3 Premissas para um método automático de combinação de ações de sintonia fina 41
4.3.1 Abordagem não-intrusiva 41

4.3.2 Execução offline 42
4.3.3 Extensão para múltiplos SGBDs e modelos de custo 43
4.3.4 Extensão para diferentes técnicas de sintonia fina 44
4.3.5 Extensão para diferentes heurísticas de sintonia fina 44
4.3.6 Resumo do capítulo 44

5 Geração automática de soluções combinadas de sintonia fina 45
5.1 Especialista 45
5.2 Comitê de especialistas 45
5.3 O problema de controle 47
5.3.1 Como combinar e propor as soluções 48
5.3.2 Coordenando múltiplas fontes de conhecimento (especialistas) na busca pela solução 50
5.3.3 Condições de parada 51
5.4 Algoritmo de geração de soluções combinadas 51
5.5 Eliminação de soluções mutuamente exclusivas 55
5.6 Exemplo de execução do Algoritmo GSC 56
5.6.1 Primeira iteração 57
5.6.2 Segunda iteração 57
5.6.3 Terceira iteração 58
5.6.4 Quarta iteração 58
5.6.5 Parada 59
5.7 Resumo do capítulo 59

6 Seleção automática de soluções combinadas 60
6.1 Algoritmo de seleção de ações combinadas 60
6.2 Modelo de Custo 63
6.2.1 Cálculo do custo e benefício de soluções combinadas 63
6.3 Resumo do capítulo 65

7 Implementação 66
7.1 O sistema multiagentes 67
7.2 Os especialistas que compõem o comitê 69
7.3 Extensão para três SGBDRs 70
7.4 Resumo do capítulo 71

8 Experimentação 72
8.1 Setup 72
8.2 Métricas 72
8.3 Benchmark 73
8.4 Cenários 75
8.4.1 Metodologia de testes 76
8.5 Resultados e Análise 76
8.5.1 Análise dos resultados no escopo dos cenários 76
8.5.2 Análise dos resultados no escopo dos modelos de consulta 78
8.5.3 Análise a partir de exemplos de soluções combinadas 81
8.5.4 Comparação entre os algoritmos GSC e GSCR 91
8.5.5 Comparação com outras ferramentas de sintonia fina 94
8.6 Resumo do Capítulo 95

9 Discussão sobre o método **96**
9.1 Extensão do método de combinação 96
9.1.1 Identificar formas de integrar as técnicas de sintonia 96
9.1.2 Eliminar soluções mutuamente exclusivas 97
9.1.3 Eliminar soluções inválidas e inexpressivas 98
9.2 Cenários de uso e atualizações na carga de trabalho 98
9.3 Resumo do capítulo 100

10 Conclusão **101**
10.1 Contribuições principais 101
10.2 Contribuições secundárias 102
10.3 Oportunidades para Trabalhos Futuros 103

A Modelos de consulta do TPC-H **115**

Lista de figuras

Figura 2.1 Diagrama simplificado do processo de modelagem de banco de dados e a integração com o processo de sintonia fina de banco de dados. Figura adaptada de Elmasri e Navathe [1] 19

Figura 4.1 Etapas do método para combinação e seleção automática de ações de sintonia fina. 40

Figura 5.1 Possíveis cenários de comitês de especialistas em sintonia fina seguindo as premissas de formação. 46

Figura 5.2 Possíveis cenários de comitês de especialistas em sintonia fina 56

Figura 6.1 Exemplo da etapa de seleção de soluções combinadas para ilustrar o modelo de custo proposto. Consideram-se duas ações para cada solução apenas para simplificá-lo. Não existe um número fixo de ações para cada solução e podem possuir diferentes quantidades. 64

Figura 7.1 Sistema multiagente de software para execução de uma tarefa de sintonia fina global 68

Figura 8.1 Distribuição do custo de execução das consultas sem nenhuma ação de sintonia fina - Cenário ORIGINAL 74

Figura 8.2 Custo de execução para cada cenário em relação ao custo original. 77

Figura 8.3 Comparação do custo de execução entre os melhores cenários testados. 77

Figura 8.4 Custo de execução para cada cenário agrupado pelas famílias de consultas Q17 e Q20. 78

Figura 8.5 Custo de execução para cada cenário agrupado por família de consultas. 79

Figura 8.6 Custo de execução para cenários ORIGINAL e IND+PIN+(VMA+IND+PIN) agrupado pelas consultas com menor custo de execução. 80

Figura 8.7 Custo de execução para cenários ORIGINAL e IND+PIN+(VMA+IND+PIN) agrupado pelas consultas com maior custo de execução. 81

Figura 8.8 Plano de execução instância da *Q4_instancia_10* - Cenário ORIGINAL - sem nenhuma ação de sintonia fina. 82

Figura 8.9 Plano de execução instância da *Q4_instancia_10* - Cenário IND+PIN+(VMA+IND+PIN). 84

Figura 8.10 Plano de execução da consulta Q11_instancia_1 no cenário ORIGINAL. O otimizador de consultas planejou 6 varreduras completas (marcadas com setas). 87

Figura 8.11 Plano de execução da consulta Q11_instancia_1 no Cenário IND+PIN+(VMA+IND+PIN) usando uma visão materializada indexada, e outros dois índices para obtenção dos dados. Apenas a tabela *nation* continua com uma varredura completa, mas ela possui 25 tuplas, logo não foram sugeridas soluções para ela. 88
Figura 8.12 Plano de execução da *Q17_instancia_2* - Cenário ORIGINAL 90
Figura 8.13 Plano de execução instância da *Q17_instancia_2* - Cenário IND+PIN+(VMA+IND+PIN) 90
Figura 8.14 Custo de execução total entre os algoritmos GSC e GSCR usando o cenário IND+PIN+(VMA+IND+PIN). Quanto menor o custo de execução, melhor. 92
Figura 8.15 Custo de execução total entre os algoritmos GSC e GSCR e o cenário IND+PIN 93
Figura 8.16 Exemplo de execução do algoritmo GSC e GSCR. 93
Figura 8.17 Ganho total de cada cenário em relação às ferramentas Dexter e PoWA. Quanto maior o ganho, melhores foram as ações de sintonia fina. 94
Figura 8.18 Custo de execução total dos melhores cenários e das ferramentas Dexter e PoWA. Quanto menor o custo, melhor o desempenho. 94

Lista de tabelas

Tabela 8.1 Estatísticas sobre o banco de dados do TPC-H usado nos experimentos. 73

Lista de abreviaturas

DBA	*Database Administrator* - Administrador de banco de dados
BnB	*Branch-and-Bound*
SGBD	Sistema Gerenciador de Banco de Dados
VMA	Visões Materlizadas
IND	Índice completo
PIN	Índice Parcial
I/O	*input/output* - entrada/saída
SQL	*Structured Query Language* - Linguagem de Consulta Estruturada
TPC	Transaction Processing Performance Council
TPC-H	TPC Benchmark H
HDD	*Hard Disk Drive*
SDD	*Solid-State Drive*
IA	Inteligência Artificial
GSC	Algoritmo de Geração de Soluções Combinadas
GSCR	Algoritmo de Geração de Soluções Combinadas Relaxado
SCC	Algoritmo de Seleção de Soluções Combinadas
TB	Terabytes
GB	Gigabytes
MB	Megabytes
kB	Kilobyte
JADE	JAVA Agent DEvelopment Framework
FIPA	Foundation for Inteligent Physical Agents
ML	Aprendizado de máquina (*machine learning*)
DL	Aprendizado profundo (*deep learning*)

1
Introdução

Há uma crescente demanda por ferramentas que automatizem tarefas complexas em sistemas computacionais. No caso particular de bancos de dados, podemos citar as tarefas relacionadas às atividades de sintonia fina (*database tuning*).

O trabalho de administração e sintonia fina de bancos de dados é complexo, exige especialização e conhecimentos fundamentais nesta área da computação. Vários sistemas foram desenvolvidos para apoiar o administrador de banco de dados (DBA - database administrator) nesta atividade, cujo objetivo principal é garantir disponibilidade e bom desempenho do banco de dados. Em particular, alguns destes sistemas permitem a realização de tarefas de sintonia fina de maneira (semi) automática [2, 3]. Em sua maioria, são sistemas desenvolvidos a partir do conhecimento de um DBA especialista, gerando algoritmos e heurísticas de manutenção do sistema de banco de dados.

O principal objetivo da sintonia fina é buscar um melhor desempenho para o banco de dados através da redução do tempo de resposta das consultas e/ou aumento da vazão (*throughput*) das transações. Para isso, realizam-se ajustes de suas configurações, parâmetros, seleção de estruturas de acesso, duplicação de estruturas físicas, sempre de acordo com a carga de trabalho executada no banco. Existem diversas propostas de ferramentas que auxiliam a sintonia fina através da automatização de estratégias para a seleção de índices e visões materializadas, particionamento, reescrita de consultas, entre outras. No entanto, tais técnicas são normalmente aplicadas isoladamente, buscando soluções locais para cada tipo de ação.

Diferentes técnicas de sintonia fina possuem diferentes estratégias para minimizar o custo de execução da carga de trabalho. Por exemplo, visões materializadas pré-processam e armazenam resultados para comandos SQL da carga de trabalho, para que no futuro não seja necessário o reprocessamento daquela mesma consulta. Já índices são estruturas de acesso que permitem acessar de forma eficiente tuplas de uma determinada tabela que satisfaçam condições de pesquisa nos atributos de chave de busca do índice. Isso evita o custo de percorrer a tabela por completo durante a execução de um comando SQL.

No entanto, as diferentes técnicas são, normalmente, geradas e selecionadas a partir de uma análise local para cada tipo de ação. Dado este cenário, propomos a hipótese desta pesquisa:

1.1 Hipótese

Ações de sintonia fina combinadas e selecionadas através de uma análise global da carga de trabalho podem gerar um benefício maior que ações individuais selecionadas localmente

1.2 O problema de pesquisa

O processo de combinação de ações de sintonia fina em banco de dados relacionais ainda não possui um método sistemático e independente das técnicas envolvidas. A questão ***como combinar*** ainda precisa ser investigada. Não há na literatura, nenhum método abrangente para a geração de ações combinadas durante uma única tarefa de sintonia fina.

1.3 Objetivo Geral

O objetivo geral desta tese é propor um método de combinação independente de técnicas, sistemático e automático para ações de sintonia fina em banco de dados relacionais.

1.3.1 Questões de pesquisa

Q1: **Como gerar combinações de ações válidas durante uma tarefa de sintonia fina global?** Dado um conjunto de técnicas de sintonia fina, existe a possibilidade de que nem todas as combinações possíveis entre elas sejam válidas de acordo com as técnicas envolvidas. Esta questão investiga como identificar soluções inválidas e eliminá-las durante o processo de combinação.

Q2: **Como selecionar ações combinadas que respeitam restrições tecnológicas dos SGBDs e os recursos computacionais disponíveis?** Recursos computacionais não são infinitos durante a tarefa de sintonia fina. Dado que as combinações podem gerar um espaço de busca muito grande, esta questão tem por objetivo estudar como selecionar um conjunto de ações combinadas que respeitam estes limites enquanto minimizam o custo de execução da carga de trabalho.

1.3.2 Objetivos Específicos

Dado as questões de pesquisa Q1 e Q2 anteriores, definimos os seguintes objetivos específicos:

OE1:**Propor um método amplo e automático para combinação de múltiplas técnicas de sintonia fina.**. Entende-se que o método proposto não deve ser específico para um conjunto de técnicas e deve prever a integração de diferentes técnicas existentes na literatura ou futuras técnicas desenvolvidas para o processo de combinação.

OE2: **Propor um método de combinação que consiga identificar e filtrar ações combinadas: (a) mutuamente exclusivas; (b) inválidas; (c) inexpressivas**. O método pode gerar teoricamente um número talvez inviável de combinações. Logo, é necessário identificar formas de diminuir o espaço de busca. As ações combinadas (a), (b), e (c) são possíveis formas de reduzir o espaço de busca.

OE3: **Selecionar globalmente ações combinadas sob restrições computacionais que minimizam o custo de execução da carga de trabalho**. Uma vez que o método é capaz de gerar as ações combinadas, é necessário filtrar o subconjunto que otimize a execução dos comandos SQL, mas que também respeite os limites computacionais envolvidos.

1.4 Avaliação

Para avaliar o método, apresenta-se uma implementação possível dos algoritmos propostos, um conjunto de cenários de teste, e uma experimentação usando três SGBDs do mercado para gerar evidências sobre a eficácia e eficiência da estratégia de combinação. Este trabalho também é comparado com duas ferramentas de sintonia fina *open-source*. Resultados evidenciaram que as ações combinadas selecionadas globalmente podem gerar melhores resultados que ações escolhidas localmente para cada tipo de técnica de forma independente.

1.5 Contribuições

As principais contribuições desta tese são:

(i) um método de combinação de ações de sintonia fina independente de técnicas, e automático capaz de:

  - gerar ações combinadas entre diferentes técnicas percorrendo o espaço de busca através de um algoritmo de força bruta, mas aplicando restrições devido à complexidade de se testar todas as alternativas.
  - selecionar através de uma estratégia global o conjunto de ações combinadas que minimizam o custo de execução da carga de trabalho e respeitam os limites dos recursos computacionais disponíveis.

(ii) Discussão sobre a geração e seleção de ações combinadas em um processo de sintonia fina global, e os principais desafios da automação destas etapas.

1.6 Organização da Tese

O Capítulo 2 apesenta conceitos e fundamentos discutidos nesta tese. O Capítulo 4 apresenta uma descrição do método de combinação de ações de sintonia fina aqui proposto. O Capítulo 3 apresenta os principais trabalhos relacionados. O Capítulo 5 descreve a etapa de geração das ações combinadas, e o Capítulo 6 apresenta a etapa de seleção das ações combinadas que minimizam o custo global de execução da carga de trabalho na presença de limites computacionais. O Capítulo 7 descreve os principais desafios e decisões de implementação do método, e o Capítulo 8 apresenta uma avaliação experimental do método proposto. O Capítulo 9 discute como estender o método e o desafio de selecionar ações de sintonia fina para cargas com atualizações. Por fim, o Capítulo 10 conclui a tese e apresenta as principais oportunidades de trabalhos futuros.

2
Conceitos Fundamentais

Antes de descrever o método para combinação de ações de sintonia fina, vale ressaltar alguns conceitos importantes para o entendimento do problema estudado aqui. A Seção 2.1 apresenta o processo de modelagem de um banco de dados, define o que é sintonia fina e descreve as diferentes técnicas usadas aqui. A Seção 2.3 apresenta o método *Branch-And-Bound*, e a Seção 2.4 discute os desafios da modelagem de um sistema multiagentes, ambos necessários para o entendimento da implementação do método de combinação aqui proposto.

2.1
Modelagem do banco de dados e a tarefa de sintonia fina

A modelagem do banco de dados é o um processo muito importante na qual define-se desde os requisitos de dados, até o esquema físico do banco que será implementado. É um processo sistemático que possui fases bem definidas [1, 4].

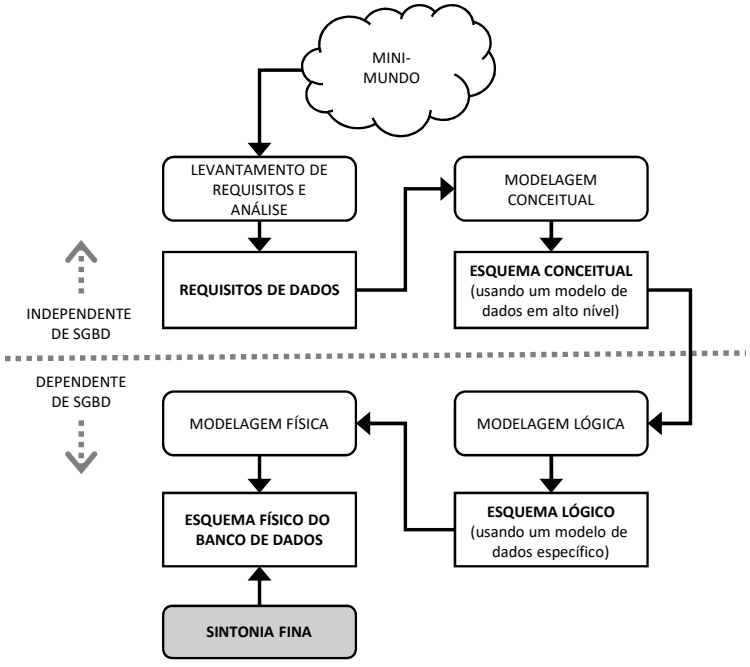


Figura 2.1: Diagrama simplificado do processo de modelagem de banco de dados e a integração com o processo de sintonia fina de banco de dados. Figura adaptada de Elmasri e Navathe [1]

A Figura 2.1 mostra o processo de modelagem de banco de dados. O primeiro passo é o levantamento de requisitos e análise sobre o "mini-mundo"que o banco de dados irá representar. Esta tarefa produzirá um documento sobre os requisitos de dados, que será usado como entrada do processo de modelagem conceitual do banco de dados.

A modelagem conceitual é a fase onde o esquema conceitual é gerado, e usado para representar em mais alto-nível uma descrição dos dados a serem armazenados, juntamente com as restrições de integridade conhecidas para serem aplicadas a estes dados. Nesta fase definem-se as entidades, seus atributos e relacionamentos. O modelo conceitual é independente de tecnologia, e é usado para discutir com o usuário os aspectos do negócio, ou "mini-mundo". Não envolve discussões quanto às estruturas de armazenamento ou tecnologias envolvidas.

O próximo passo é, a partir do esquema conceitual, realizar a modelagem lógica e gerar o esquema lógico do banco de dados. Nesta fase, decide-se qual será o modelo de dados usado para representação, como modelo relacional, orientado a objetos, entre outros. É o primeiro passo na direção da tecnologia envolvida na construção dos SGBDs, mas ainda no nível estrutural.

Por fim, o esquema lógico é usado durante a modelagem física do banco. Aqui realiza-se o mapeamento do esquema lógico nas estruturas físicas oferecidas pelo SGBD escolhido. O esquema físico é específico para cada SGBD e é onde consideram-se as tecnologias disponíveis e quais benefícios cada tecnologia pode trazer. São escolhidos também métodos de acesso como quais índices serão criados, tabelas, particionamentos, e outras decisões que podem influenciar na obtenção e atualização dos dados.

Após as três fases de modelagem e a implementação do banco de dados, este está pronto para executar a carga de trabalho. De maneira geral, podemos considerar uma carga de trabalho composta por (i) uma lista de consultas com sua frequência, (ii) uma lista das atualizações (inserções, modificações e exclusões) nas instâncias de dados e suas frequências, (iii) os dados armazenados [4].

A atividade de sintonia fina (*tuning*) é o processo de refinar, ou ajustar, o esquema físico do banco de dados e os parâmetros e configurações do SGBD de acordo com os padrões da carga de trabalho e as características do SGBD, para que se possa melhorar o desempenho [4, 2]. Esta tarefa pode ser necessária tanto por uma falta de conhecimento de características da carga de trabalho durante a modelagem física do banco de dados, quanto por mudanças naturais destas características ao longo do tempo.

Administradores de banco de dados realizam ajustes no projeto físico do

banco de dados através da seleção de estruturas de acesso (índices ou visões materializadas), duplicação de estruturas físicas, determinação dos objetos a serem particionados e seus respectivos tipos de particionamento, ajustes em configurações do SGBD, e outros. Sempre de acordo com a carga de trabalho executada [5][4].

Existem propostas de ferramentas para sintonia fina que realizam seleção de índices e visões materializadas, particionamento automático, reescrita de consultas SQL, ajustes nas configurações do SGBD (Ex.: controle de *buffer*), otimização de planos de execução, entre outras [3, 2, 6].

2.2 Técnicas de sintonia fina

Em um sistema de banco de dados, tabelas são armazenadas em disco como uma sequência de páginas, tipicamente de tamanhos entre 4KB a 64KB. Para cada tupla em uma tabela é estabelecido um identificador de registro (*Record id - RID*), que identifica a página que contém a tupla. Logo, as páginas são as unidades de entrada/saída (*input/output - (I/O)* transferidas entre a memória secundária e memória principal [3].

Entre as diferentes estratégias de sintonia fina propostas na literatura, discutimos aqui três em especial: índices, índices parciais, e visões materializadas. As três possuem o objetivo em comum de reduzir o custo de I/O, e por sua vez, reduzir do número de páginas lidas durante a execução de comandos SQL. Para alcançar este objetivo, cada uma aplica uma estratégia diferente.

2.2.1 Índices

Índices são uma parte importante dos SGBDs, que podem melhorar drasticamente a execução de consultas. Um índice em um SGBD é uma estrutura de dados que organiza tuplas de dados de maneira persistente para otimizar determinados tipos de operações de acesso a estes dados. Um índice permite acessar de forma eficiente todas as tuplas que satisfaçam condições de pesquisa nos atributos de chave de busca do índice [4].

As estruturas de índice são arquivos adicionais no disco que fornecem caminhos de acesso secundários, e maneiras alternativas de acessar tuplas sem afetar o armazenamento físico dos arquivo de dados no disco. Eles permitem acesso eficiente a tuplas com base nos atributos de indexação usados para construir o índice. Basicamente, qualquer campo da tabela pode ser usado para criar um índice, e vários índices em diferentes atributos - assim como índices em vários atributos - podem ser construídos na mesma tabela [1].

Um índice é definido sobre uma sequência de colunas de uma tabela e serve como uma estrutura de acesso para obter tuplas que satisfazem certos predicados. Uma chave de busca é uma sequência de atributos. O índice define um mapeamento entre os valores desta sequência de atributos e suas tuplas correspondentes. Logo, eles fornecem um caminho eficiente para localizar e acessar os dados de maneira mais rápida, além de reduzirem a sobrecarga de busca a um pequeno conjunto de tuplas. [7].

Uma variedade de índices é possível, cada um deles usa uma estrutura de dados específica para acelerar a pesquisa. Para localizar uma ou mais tuplas na tabela com base em uma condição de pesquisa em um atributo de indexação, o índice é pesquisado, o que leva a ponteiros para um ou mais blocos de disco em que as tuplas necessárias estão localizadas [1].

Entre os tipos comuns de índices, estão os baseados em arquivos ordenados (índices de nível único), e índices que usam estruturas de dados em árvore (índices multiníveis, como B+trees) para organizar o índice. Índices também podem ser construídos com estruturas *hash*, ou vetores de *bits* chamados de índices *bitmap* [1].

2.2.2 Índices completos

Um índice completo (IND) é o índice tradicional definido na literatura e implementado nos SGBDs. Na literatura ele é chamado apenas índice, mas usa-se o termo índice completo nesta pesquisa para distingui-lo do índice parcial.

Um índice completo pode ser primário ou secundário. O índice primário é aquele que indexa o atributo chave, e no qual a ordem das tuplas na tabela subjacente é idêntica à ordem das chaves contidas nas folhas do índice. Já um índice secundário é aquele que não há relação entre a ordem das chaves nas folhas do índice e das tuplas armazenadas na tabela [6]. Neste trabalho, são descritos algoritmos para a criação apenas de índices secundários por gerar um número maior de alternativas de indexação. Enquanto a ordenação física das tuplas de uma tabela não precisa refletir a ordenação das chaves de um índice secundário, então é possível definir um número arbitrário de índices deste tipo.

2.2.3 Índices parciais

Os índices parciais (PIN) são índices que permitem definir um subconjunto das tuplas de uma tabela, através de uma expressão condicional [8]. Em alguns casos, os índices parciais podem melhorar o desempenho da execução das consultas no banco de dados através da redução tanto da quantidade de

leituras lógicas como do custo de processamento. Geralmente, os índices parciais são usados nos casos em que há uma diferenciação dos dados ativos e não ativos ou por conta de existirem termos específicos de busca em certos conjuntos de dados.

Por exemplo, considere uma tabela *vendas* com, entre outros, o atributo *data_venda*. Pode-se definir um índice parcial como:

```
1 create index pin_data_venda
2 on venda(data_venda)
3 where data_venda between '01/01/2018' and '01/01/2019';
```

Note que o atributo de restrição da cláusula *where* restringe o índice *pin_data_venda* às tuplas onde a data da venda entre '01/01/2018' and '01/01/2019'. Neste exemplo, caso a tabela *vendas* possua vendas de muitos anos anteriores, o índice parcial indexaria um número menor de tuplas que um índice completo. Consequentemente o PIN teria menor tamanho, e poderia trazer um benefício maior que o IND pois o custo de obter os dados do índice da memória secundária seria maior, e o espaço de busca (tuplas indexadas) no IND também seria maior. Por outro lado, o *pin_data_venda* não seria útil para consultas em tuplas fora da faixa de valores indexada, o que provavelmente acarretaria em uma varredura completa da tabela *vendas* e um custo alto de execução para o comando SQL.

2.2.4 Visões materializadas

Uma visão é uma tabela cujas tuplas não são armazenadas explicitamente no banco de dados, mas são calculadas conforme for necessário, com base em uma definição de uma consulta. A visão pode ser usada exatamente como uma tabela na definição de novas consultas ou visões. Entre as suas utilidades estão fornecer independência lógica de dados e criar simplificações de consultas complexas e frequentemente usadas [4].

Já uma Visão Materializada (VMA) é uma visão que possui o seu resultado previamente calculado e armazenado para uso posterior. Quando uma consulta é realizada em uma visão materializada, a consulta é executada diretamente no resultado armazenado da VMA. Esta técnica pode trazer um ganho significativo, pois a consulta não precisa recalcular o seu resultado durante a execução [4].

Visões materializadas acarretam em custos. Não é necessário ter apenas espaço em disco suficiente mas, também, os custos de criação e manutenção. Quando uma tabela é atualizada, o conteúdo da visão materializada se torna desatualizado. É necessário então selecionar quais visões materializadas trazem

o maior benefício e os menores custos possíveis, para uma dada carga de trabalho [9].

Pode-se formalizar a escolha do conjunto de visões materializadas consideradas viáveis pela seguinte definição: Dado um esquema de um banco de dados R, armazenado em espaço D e uma carga de trabalho de consultas Q, deve-se escolher um conjunto de consultas V sobre R para ser materializado de forma que o somatório do tamanho dos elementos de V seja menor que D. [9].

Além da questão de viabilidade, deve-se considerar quais VMAs podem ser úteis para melhorar o desempenho de um sistema de banco de dados. A escolha do conjunto de consultas, cujo custo de execução da carga de trabalho é minimizado, dado a uma restrição de espaço em disco e um custo de manutenção é um problema com complexidade exponencial [4]. Não é possível materializar todas as possíveis visões a melhorar o desempenho por pelo menos dois motivos: (i) o espaço em disco necessário para materializar todas as opções pode inviabilizar esta escolha e (ii) o custo de manter as visões materializadas atualizadas pode ser proibitivo [9].

2.3 Branch-and-Bound

O método *Branch-and-bound* (BnB) baseia-se na ideia de desenvolver uma enumeração inteligente das ações combinadas à solução ótima de um problema. O termo *branch* refere-se ao fato de que o método efetua partições no espaço de soluções. O termo *bound* ressalta que a prova da otimalidade da solução utiliza-se de limites calculados ao longo da enumeração.

O BnB enumera sistematicamente todas as soluções candidatas e permite que grandes conjuntos de candidatos inválidos sejam descartados. Logo, apenas uma fração das soluções factíveis é realmente examinada. O método BnB consegue gerar soluções para problemas complexos reduzindo o espaço de busca, onde normalmente pode ser inviável testar todas as soluções [10].

O BnB é uma técnica de ampla aplicação. A ideia geral é sujeita a inúmeras adaptações e estratégias de implementação, mas de forma geral quatro componentes caracterizam o algoritmo:

1. **Uma regra de branching**: como um estado (também chamado de nó em alguns trabalhos) origina seus "filhos";

2. **Uma regra de seleção**: indica a ordem em que os estado ativos serão expandidos. Durante a execução os estados ativos são colocados em uma lista. A ordem em que são escolhidos determina a forma de busca. Quando

o estado é expandido ele é retirado da lista. Alguns tipos de buscas possíveis são:

a) **Profundidade**: nesta forma de busca a expansão dos estados se dará da forma LIFO (*Last in, first out*). Ou seja, o estado a ser expandido será o último acrescentado à lista de ativos.

b) **Largura**: nesta forma de busca a expansão dos estados se dará da forma FIFO (*First in, first out*). Ou seja, o estado a ser expandido será o primeiro acrescentado a lista de ativos.

c) **Mais promissor**:nesta forma de busca a expansão dos estados se dará utilizando algum critério (heurística) que indique que o estado escolhido levará à melhor solução. Por exemplo escolher o estado de menor valor para a função objetivo, para problemas de minimização.

3. **Uma regra de eliminação**: é o critério adotado para descartar estados ativos. Testes de *lower-bound* e dominância são exemplos deste critério.

4. **Uma condição de término**: para problemas de combinação, como o proposto aqui, é a busca exaustiva. O problema só é terminado quando foram feitas todas as tentativas de expansão.

2.4 Sistema multiagentes

Um agente de software é um sistema computacional capaz de ações autônomas com o objetivo de alcançar os objetivos para que foi modelado. Suas ações sofrem influência do ambiente em que o agente atua e a saída é determinada pelos efeitos que o agente causa em seu ambiente [11].

Agentes possuem duas características fundamentais para este trabalho. Autonomia e inteligência. A autonomia diferencia agentes dos programas tradicionais por ser capaz de seguir seus objetivos, e adaptando-se à mudanças no ambiente automaticamente. O grau de autonomia é definido pelo engenheiro de software durante a modelagem dos comportamentos do agente, e é sensível ao domínio que esta solução é aplicada.

Áreas sensíveis como mercado financeiro ou ambientes que envolvam risco à vida humana, por exemplo, apresentam limitadores naturais ao nível de automação. No entanto, nestas mesmas áreas atividades como monitoramento e notificação possuem alto potencial na aplicação de agentes autônomos [12].

Na área de sintonia fina de banco de dados, existem trabalhos onde o uso de agentes autônomos tem sido usadas para implementar propostas

automáticas de monitoramento da carga de trabalho e ajustes no esquema físico do banco de dados [13, 14, 15, 16, 17]

Já a inteligência de um agente é formada por um ou mais dos seguintes componentes: a) base de conhecimento, b) capacidade de raciocínio baseado em sua base de conhecimento, c) capacidade de aprender ou se adaptar a mudanças ocorridas no ambiente. A inteligência resultante destes componentes permite, por sua, vez aumentar a capacidade de automação do agente [18, 11, 19].

No contexto deste trabalho, cada agente acessa uma base de conhecimento definida como modelo de custos, possui capacidade de raciocínio sobre esta base, uma vez que usa ela para gerar ações de sintonia fina, e também possui a capacidade de se adaptar às mudanças ocorridas no ambiente. Uma vez que o estado corrente do banco de dados, e ações geradas por outros agentes podem influenciar as soluções propostas por determinado agente, considera-se um tipo de adaptação. Apesar de modelos de custo (a base de conhecimento) e algoritmos apontarem para determinadas soluções como as melhores, os agentes conseguem incluir na avaliação informações sobre ações concorrentes e o estado atual do banco.

2.5 Resumo do Capítulo

Este Capítulo apresentou brevemente os principais conceitos utilizados para descrever o método de combinação proposto. Foi apresentado o processo de modelagem de um banco de dados, a definição de sintonia fina, e técnicas de sintonia fina utilizadas aqui. Descreveu-se também o método branch-and-bound e sistemas multiagentes. O Capítulo a seguir apresenta os principais trabalhos relacionados a esta pesquisa.

3
Trabalhos relacionados e revisão da literatura

Sintonia fina de banco de dados relacionais possui uma literatura extensa. Muitos algoritmos e heurísticas de sintonia fina já foram propostos na literatura. Este Capítulo apresenta os principais trabalhos relacionados a esta tese. A Seção Seção 3.1 apresenta estratégias diretamente relacionadas ao método aqui proposto. A Seção 3.2 apresenta uma revisão bibliográfica que serviu de apoio para o desenvolvimento do método, e a Seção 3.3 apresenta ferramentas semi-automáticas e automáticas de sintonia fina que implementam algoritmos de sintonia fina.

3.1
Principais Trabalhos Relacionados

Agrawal et. al. [20] propõe um método automático para geração e seleção de dois tipos de ações: índices e visões materializadas. O trabalho propõe a combinação das duas estruturas de acesso através de duas fases. Na primeira as ações para cada uma das técnicas são geradas de forma independente, e na segunda as ações geradas são filtradas de acordo com duas políticas de seleção possíveis: MVFIRST e INDFIRST.

Usando a MVFIRST, o algoritmo seleciona primeiro as visões materializadas que, segundo um modelo de custo próprio, trarão os melhores benefícios e respeitarão o espaço de armazenamento disponível. Em seguida, os melhores índices são selecionados de acordo com o espaço em disco restante.

Já na estratégia INDFIRST, apenas a ordem de seleção das técnicas é inversa à MVFIRST. Usando INDFIRST a seleção de índices é feita primeiro, e visões materializadas são selecionadas somente após a seleção de todos os índices considerados bons candidatos para criação no banco.

Outro trabalho que explora a combinação de índices e visões materializadas é Kimura et. al. [21]. Ele apresenta um *framework* chamado CORADD, que explora correlações entre atributos para recomendar visões materializadas e índices. A ferramenta avalia a carga de trabalho e encontra um conjunto de visões materializadas e índices para serem criados de forma combinada de acordo com um limite de armazenamento disponível para ações de sintonia fina.

Durante a fase de geração dos índices e visões materializadas, Kimura et. al. utilizam a mesma estratégia de Agrawal et. al., e geram as ações de sintonia fina a partir de algoritmos independentes para cada técnica, tratando a combinação apenas na fase de seleção.

Na fase de seleção, Kimura et. al. agrupa os comandos SQL da carga de trabalho através do algorítimo *k-means* [22] e, para cada *cluster*, seleciona as melhores visões materializadas, seguidas de um conjunto de índices. O número de estruturas de acesso selecionadas para cada um é obtido através de uma função que calcula pesos para cada *cluster* de acordo com o custo de execução total. Grupos de comandos que possuem custos de execução maiores que outros recebem um número maior de ações de sintonia fina, proporcional ao seu peso calculado.

Existe também um *framework* para sintonia fina chamado *DBX*, proposto por [6] e estendido posteriormente [16]. É uma arquitetura baseada em agentes de software que é capaz de gerar ações de sintonia fina para múltiplas técnicas. Na versão atual gera ações para índices completos e visões materializadas. Apesar da capacidade de gerar múltiplos tipos de ações, a modelagem proposta tanto para os métodos de geração quanto seleção não consideram combinação de ações. Na fase de geração, o algoritmo gera as ações independentes. Na fase de seleção, todas as ações geradas são ordenadas pelo benefício de cada ação, e as top-k ações são selecionadas.

Por fim, existe outro *framework* chamado Outer-Tuning, proposto por Almeida [23], e também estendido posteriormente [24, 25, 26]. Foi realizado dentro do grupo de pesquisa em bancos de dados e bioinformática BioBD da PUC-Rio. Almeida [10] propõe uma ontologia de sintonia fina na qual é possível incluir novas formas de sintonia fina, e teoricamente, combinar tais técnicas. O trabalho tem por objetivo fundamentar o domínio de sintonia fina e absorver múltiplos tipos de ações em uma única base de conhecimento (a ontologia). A ontologia de sintonia fina contempla uma ontologia de domínio, que define os conceitos envolvidos na tarefa de sintonia fina, e uma ontologia de tarefas, onde são instanciadas as heurísticas para geração e seleção de ações.

Na ontologia proposta para o Outer-Tuning, as fases de geração e seleção são implementadas através da instanciação de um conjunto de heurísticas na ontologia de tarefas. A ontologia possui, na versão corrente, heurísticas para criação e seleção de visões materializadas e índices completos de formas independentes. Teoricamente, é possível gerar ações combinadas, mas não existe nenhuma heurística implementada que considere tais casos.

Já esta tese propõe um método de combinação abrangente e automático para ações de sintonia fina em banco de dados relacionais. Estuda-se aqui,

como combinar diferentes técnicas e respeitar os limites de recursos computacionais disponíveis.

Em relação a este trabalho, as pesquisas de Agrawal et. al., Kimura et. al., não apresentam as mesmas contribuições porque propõe combinações específicas para duas técnicas de sintonia fina. Além disso, nenhum deles apresenta um algoritmo para geração de ações combinadas para as técnicas envolvidas. A combinação, considerada apenas na fase de seleção, não é fruto de um método sistemático de exploração das possíveis combinações entre as técnicas, como proposto aqui.

Já em relação ao DBX, as ações também são geradas por processos (ou agentes neste caso) independentes que não consideram a combinação das técnicas nesta fase. E na fase de seleção, apesar de uma heurística de cálculo de benefício sofisticada, o conjunto final é uma seleção de top-k melhores ações para a carga de trabalho. Não faz parte da proposta do *framework* um método para geração e avaliação de ações combinadas.

No caso do Outer-Tuning discute-se a possibilidade de gerar ações combinadas desde a publicação inicial por Almeida [23]. Porém não é apresentado um método de combinação sistemática de ações. Entende-se que, teoricamente, é possível combinar ações explorando uma linguagem declarativa como usada na ontologia e um motor de regras. Porém, não se define *como* fazer isso, isto é um método de combinação capaz de usar as duas técnicas atualmente usadas na ontologia.

Desta forma, não é do nosso conhecimento nenhuma pesquisa ou ferramenta que proponha um método de combinação sistemático e abrangente para ações de sintonia fina em banco de dados relacionais.

A seguir, apresenta-se trabalhos que não são diretamente relacionados com esta proposta, mas que também são importantes como parte da revisão da literatura sobre sintonia fina de banco de dados.

3.2 Algoritmos para sintonia fina local

Um grande número de pesquisas que apresentam algoritmos de geração e seleção de ações de sintonia fina na literatura. São trabalhos que propõem diferentes estratégias de construção e ranqueamento de ações para diferentes tipos de técnicas. Nenhum deles está diretamente relacionado com a pesquisa desta tese por tratarem a geração e/ou seleção das técnicas independentemente, sem necessariamente combiná-las com outras técnicas. Porém, considera-se interessante listar as principais estratégias já que são todas, teoricamente, passíveis de extensão pelo método de combinação proposto aqui.

Visões materializadas: algoritmos gulosos [27, 28, 29] onde a cada iteração se seleciona a VMA de maior benefício. O processo se repete até que um número predeterminado de VMAs sejam selecionadas, ou se todo o espaço em disco disponível seja consumido. Existem também heurísticas randômicas onde cada ação de sintonia fina é considerada um estado com um custo associado, e o espaço de busca é explorado através de passos randômicos usando métodos como *hill-climbing* [30, 31], *simulated annealing* [32, 33, 34] e *shuffled frog leaping* [35]. Heurísticas genéticas (ou evolucionárias) [36, 37] exploram o espaço de busca randomicamente, baseado no conceito genético que simula o processo de evolução biológica na natureza, onde o genoma mais adaptado (melhor solução) sobrevive após algumas gerações (iterações).

Índices Completos: Existem diferentes estratégias para adquirir um conjunto satisfatório de índices para uma determinada carga de trabalho. Algumas auxiliam a testar diferentes índices através de simulações interativas e produzir informações para a tomada de decisão [38], porém com uma abordagem manual, uma vez que o DBA precisa propor tais índices. O principal problema de tais abordagens é que se trata de métodos subjetivos à experiência do DBA e não escalam para grandes bancos de dados com complexas cargas de trabalho. Logo, abordagens automáticas para a geração e seleção de índices são populares na literatura.

Muitos estudos propõem de forma automática índices com apenas uma coluna [39, 40, 41], enquanto outros trabalhos consideram a habilidade da maioria dos SGBDs modernos de usar índices em múltiplas colunas usando diferentes estratégias: clusterizando atributos similares em um mesmo índice [42, 43, 44, 3, 2], utilizando a frequência em que os atributos são referenciados juntos na carga de trabalho [45] e iterando a partir de atributos simples para atributos compostos [46].

Em particular, o trabalho de Chaudhuri [44] é importante porque formaliza a seleção de índices como um problema de otimização e demonstra-se que, tanto a escolha de uma configuração ótima de índices secundários como de índices primários, são problemas NP-difíceis.

Índices Parciais: O trabalho de Stonebraker de 1989 [8] chamou a atenção para a aplicação dos índices parciais em SGBDs relacionais. O artigo discute como este tipo de estrutura pode ser aproveitada em ambientes onde existe uma diferenciação entre os dados ativos e não ativos. Segundo o autor, se existe um maior interesse por um subconjunto dos dados, não indexar todas as tuplas pode diminuir os custos de armazenamento, atualização e execução das consultas. Lembra ainda que o tamanho do índice influencia nos custos de execução da consulta, e mesmo estruturas otimizadas como Árvore B+

[1] ainda possuem um alto custo de leitura e manutenção. Outros trabalhos seguiram este questionamento [47, 48, 49, 50, 51]. Em particular, o trabalho de Dominguez [52] apresenta uma proposta para a geração automática de índices parciais através de uma mineração de atributos interessantes e suas respectivas faixas de valores indexáveis. Este trabalho se difere dos demais por propor uma combinação com índices completos. Vale ressaltar que ele apresenta um algoritmo que identifica em quais casos um índice completo é melhor que múltiplos índices parciais. Entendemos que a pesquisa de Dominguez pode gerar ações combinadas entre índices parciais e índices completos. Entretanto não se trata de um método de combinação abrangente e é específico para as duas técnicas envolvidas.

3.3 Técnicas de aprendizado de máquina

Técnicas de aprendizado de máquina (*Machine Learning - ML*) e aprendizado profundo (*deep learning - DL*) vêm sendo recentemente estudadas para a tarefa de sintonia fina de banco de dados, em particular para a seleção de parâmetros de configurações do SGBD.

A seleção de parâmetros é uma técnica que permite uma integração mais fácil com ML e DL do que, por exemplo, a geração e seleção de estruturas de acesso, por exemplo. Isso porque os parâmetros a serem ajustados possuem valores numéricos que podem ser representados no formato de uma tabela de atributos (*features*). Basicamente, estas técnicas usam esta tabela para treinar modelos em tarefas de regressão e predizer os melhores valores de cada parâmetro do SGBD.

Zheng [53] apresenta uma modelagem de redes neurais para automaticamente selecionar os melhores parâmetros para o SGBD. Trata-se de uma arquitetura que constantemente atualiza a rede neural com estatísticas do banco e os valores correntes dos parâmetros. Pelo histórico de consultas, a ferramenta consegue predizer sazonalidades na carga de trabalho e propor modificações nos parâmetros do SGBD.

Existe ainda uma ferramenta chamada OtterTuning [54, 55] que se destaca ao aplicar aprendizado de máquina para encontrar parâmetros de configuração. A principal contribuição é o modelo *online* para adequar os parâmetros para diferentes cargas de trabalho. O OtterTuning reutiliza dados de tarefas de sintonia fina anteriores para treinar modelos e aplicar o conhecimento adquirido a novos cenários. Esta ferramenta executa três passos principais: i) seleciona os parâmetros de configuração mais importantes; ii) mapeia valores de parâmetros aprendidos anteriormente aos novos parâmetros que deverão

ser sintonizados; e iii) recomenda novas configurações que podem beneficiar o objetivo da tarefa de sintonia (ex. vazão ou latência). A ferramenta possui um alto nível de automação, com execução automática após o treinamento dos modelos. Foi testada em três SGBDs (MySQL, PostgreSQL e Vector), o que apesar de não ser explícito nos artigos, sugere ser desacoplada do código fonte do SGBD. Por fim, propõe apenas ações locais, uma vez que gera ações de sintonia considerando apenas parâmetros, e nenhum outro tipo de ação de sintonia fina é considerado durante o processo de geração e seleção das ações. Esta mesma estratégia ainda foi estendida também para predizer parâmetros em ambientes virtualizados [56].

3.4 Ferramentas de sintonia fina automática proprietárias

São listadas aqui as principais ferramentas proprietárias e de código aberto. Diferente da abordagem proposta aqui para ferramentas de sintonia fina (Seção 4.3), elas não podem ser usadas em múltiplos SGBDs e são intrusivas, uma vez que estão integradas à implementação dos seus respectivos SGBDs. No entanto, são ferramentas importantes pelas abordagens (algumas vezes) automáticas, ou pela capacidade de gerar e avaliar ações combinadas com alta eficácia.

3.4.1 Automatic SQL Tuning (Oracle)

O Automatic SQL Tuning [57] é uma função nativa do SGBD Oracle (a partir da versão 10G) que permite ao DBA executar uma carga de trabalho usando dois modos:

Normal mode: o otimizador compila o comando SQL e gera o plano de execução considerando suas limitações de tempo tradicionais. A enumeração de possibilidades do plano é reduzida, e considera-se suficiente apenas um plano "bom", e não o ótimo para cada comando SQL.

Tuning mode: o otimizador de consultas executa análises adicionais para checar quando o plano executado no modo normal pode ser melhorado. Neste modo, o resultado não é um plano de execução, mas um conjunto de ações de sintonia fina que são executadas automaticamente no banco de dados e reportadas posteriormente ao usuário.

A função Automatic SQL Tuning utiliza-se do componente SQL Tuning Advisor para gerar as ações no Tuning mode, que por sua vez propõe dois tipos de ações: índices completos e reescrita de comandos SQL. No caso do IND, eles são gerados, avaliados e executados automaticamente no banco de

dados. A reescrita de consulta também é transparente ao usuário e realizada pelo otimizador. Por se tratar de uma ferramenta proprietária, o código fonte do SQL Tuning Advisor não está disponível para consulta e a documentação se limita a descrever o uso e seus potenciais benefícios. Não foi possível encontrar detalhes sobre os algoritmos implementados para a geração das ações de sintonia fina, mas não faz nenhuma referência a combinação de ações entre os dois tipos de ações de sintonia fina que a ferramenta suporta. Durante testes empíricos, também não foi possível identificar nenhum caso onde um único comando SQL tivesse mais de uma ação.

3.4.2 Oracle Autonomous Database

A Oracle lançou em 2018 um serviço chamado Oracle Autonomous Database, onde alega que *"o ciclo completo da gerência de banco de dados é completamente automatizado"* [58].

Segundo a documentação, o Autonomous Database é um serviço de SGBD em nuvem que aplica técnicas de aprendizado de máquina para automatizar tarefas de gestão do banco de dados tradicionalmente realizadas por um DBA humano. Ele usa a versão 18G do Oracle Database e uma infraestrutura de virtualização de *hardware* própria que, segundo a empresa, foi desenvolvida especificamente para prover serviços de SGBD no modelo PaaS (Plataform as a Service).

O papel do DBA utilizando o Autonomous Database se restringe a definir e monitorar políticas de uso, assim como os recursos consumidos que são cobrados em um modelo *pay-per-use*. Isso leva à principal diferença em relação à proposta desta tese. Assim como outras ferramentas proprietárias, os algoritmos utilizados e detalhes da abordagem são protegidos e não acessíveis, mas o Autonomous Database é um serviço de banco de dados apoiado na elasticidade que a infraestrutura de nuvem é capaz de fornecer.

No conceito de sintonia fina de banco de dados que considera-se aqui, o aumento de recursos de *hardware* não é considerado como uma ação de sintonia fina. Entende-se o poder da elasticidade proporcionada por uma arquitetura virtualizada, e que para aplicações de missão crítica são fundamentais. Porém, é de interesse desta pesquisa explorar os limites (e combinações) das técnicas dentro de um ambiente de *hardware* fixo. Em outras palavras, aumentar memória primária ou processamento são opções, mas no contexto desta tese explorar apenas aquelas técnicas que não alteram as configurações de *hardware*. Ou seja, adquirir recursos de hardware é opção apenas quando fica claro que não há como melhorar mais.

3.4.3
SQL Server Automatic Tuning

O SQL Server 2017 foi lançado com um conjunto de funcionalidades chamado Automatic Tuning [59]. Ele é responsável por identificar e notificar o DBA quando a carga de trabalho apresenta algum problema de desempenho, seguido por um conjunto de ações de sintonia fina. As técnicas usadas são índices e reescrita de consultas.

O Automatic Tuning também foi estendido para o serviço Azure SQL Database, um serviço de SGBD em nuvem fornecido pela Microsoft. Na versão para o Azure, o *Automatic Tuning* teve sua automação ampliada, e realiza a gerência de índices de forma automática e transparente para o usuário. Ele consegue criar e excluir índices de acordo com variações na carga de trabalho.

No caso do SQL Server Automatic Tuning é difícil avaliar até que ponto é comparável a este trabalho. Entende-se que ele consegue propor ações de diferentes técnicas (índices, reescrita de consultas) para a carga de trabalho, mas não podemos confirmar se estas ações são combinadas pelos algoritmos aplicados. Em experimentos empíricos, a ferramenta não sugeriu nenhuma solução onde duas técnicas diferentes são aplicadas a um mesmo comando SQL, mas não temos evidências ou informações suficientes para descartar esta hipótese. Porém, mesmo que ele tenha algoritmos para isso, não invalidaria principal contribuição desta tese: um método abrangente de combinação e seleção de ações de sintonia fina independente.

3.5
Ferramentas de sintonia fina local com código aberto

Descrevemos aqui duas ferramentas de sintonia fina que realizam sintonia fina local, o Dexter e Powa. Ambas são ferramentas de código aberto para o PostgreSQL, e estão descritas aqui porque, durante a Seção de experimentos, compara-se o desempenho delas como sendo estratégias de sintonia fina locais, com ações combinadas propostas pelo método de sintonia fina global.

3.5.1
Dexter

Dexter é uma ferramenta para geração automática de índices completos para o PostgreSQL 9.0 ou superior [60]. É uma ferramenta de código aberto. Ela utiliza uma extensão do PostgreSQL para a geração e avaliação de índices hipotéticos chamada HypoPG [61]. Esta extensão gera índices hipotéticos dentro da metabase do PostgreSQL e coleta estatísticas sobre o possível benefício destes índices para a carga de trabalho.

O Dexter gera índices hipotéticos a partir de um algoritmo próprio baseado na frequência em que os atributos das tabelas são referenciados nas cláusulas *where* dos comandos SQL. De modo geral, a ferramenta captura o plano de execução do comando SQL e propõe índices para atributos de filtragem usados no plano. A extensão HypoPG é, então, usada como modelo de custo para avaliar o benefício hipotético dos índices candidatos. Por fim, todos os índices com benefício maior que zero são sugeridos.

3.5.2 PoWA

O PoWA (*PostgreSQL Workload Analyzer*) é uma ferramenta de sintonia fina que coleta informações a partir de extensões instaladas no SGBD e propõe a criação de índices [62]. Suporta PostgreSQL versão 9.4 ou superior.

Possui algoritmo próprio para a sugestão de índices e instala um conjunto de extensões para coleta e análise da carga de trabalho. Entre elas, uma extensão chamada PoWA Archivist para a coleta de comandos SQL da carga de trabalho a serem analisados. Ela permite realizar sintonia fina *online*, adaptando o conjunto de índices propostos às sazonalidades da carga de trabalho utilizada. O PoWA, assim como o Dexter, utiliza a extensão HypoPG como modelo de custo para a avaliação dos índices propostos. A principal diferença entre o PoWA e Dexter está no processo de seleção dos índices. Enquanto o Dexter seleciona todos aqueles que trazem algum benefício para a carga de trabalho, o PoWA só sugere os índices que o custo de criação seja menor que o benefício acumulado do índice. Logo, tende a sugerir menos índices que o Dexter.

4
Método para combinação e seleção automática de ações de sintonia fina

O aumento da complexidade das aplicações de banco de dados, aliado ao aumento vertiginoso do volume de dados gerados e armazenados tem impulsionado o desenvolvimento de técnicas de sintonia fina automáticas para facilitar a tarefa de manutenção e melhora da performance do banco de dados. [63].

Este capítulo apresenta um método original para combinar diferentes ações de sintonia fina em banco de dados relacionais. Dada uma carga de trabalho, o resultado deste método é um conjunto de ações combinadas que minimizam o custo de execução da carga de trabalho sob restrições computacionais. Antes da descrição do método na Seção 4.2, a Seção 4.1 apresenta uma terminologia que será usada para explicá-lo. A Seção 4.3 discute premissas desejáveis para um método automático de combinação de ações de sintonia fina e quais foram adotadas para o método aqui proposto.

4.1
Definições

Sintonia fina de banco de dados relacionais é uma linha de pesquisa com uma extensa literatura, porém poucos trabalhos tratam da sintonia fina global. Algumas definições ou termos podem ser ambíguos, e assim registrou-se aqui algumas das terminologias para explicar o método e seus significados no contexto desta tese.

4.1.1
Solução combinada

Uma **solução combinada** é um conjunto de duas ou mais ações de sintonia fina que se complementam, possuem uma ordem de execução, e que foram selecionadas através de uma análise global da carga de trabalho, com o objetivo de diminuir o custo de execução de um conjunto específico de comandos SQL.

Entendemos que pode existir uma diferença entre solução combinada e uma solução composta de duas ou mais ações de um único tipo selecionadas

e que se completam, mas não necessariamente interagem entre si. No escopo desta tese, trataremos soluções compostas também como soluções combinadas, uma vez que os algoritmos propostos não necessitam desta distinção semântica.

4.1.2 Solução inexpressiva

Uma **solução inexpressiva** no contexto deste trabalho é uma solução que a) é gerada para beneficiar um comando SQL com um custo de execução baixo em relação ao custo total da carga de trabalho, ou b) aplica uma estrutura de acesso em uma tabela com relativamente poucas tuplas quando comparadas a outras tabelas do banco de dados.

As soluções classificadas aqui como inexpressivas são desprezadas pelo método para reduzir o espaço de busca. Dadas as restrições computacionais, o algoritmo proposto descarta estas soluções, prioriza consultas com alto impacto na carga de trabalho e tabelas com alto custo de realizar varreduras completas. Além disso, estruturas de acesso em tabelas com tamanho relativamente pequeno são normalmente ignoradas pelo otimizador de consultas durante a geração do plano de execução. De modo geral, o otimizador opta por uma varredura completa em tabelas pequenas com posterior armazenamento dos dados no *buffer*, em detrimento ao uso de estruturas de acesso.

4.1.3 Soluções mutuamente exclusivas

Duas soluções são **mutuamente exclusivas** quando tentam diminuir o custo de execução do mesmo comando SQL, e caso implementadas, forçam o otimizador de consultas a sempre descartar uma das soluções durante a execução do comando SQL.

Por exemplo, considere uma dada consulta Q sobre uma tabela T, uma heurística hi de índices que sugere um índice i na coluna c de T, e uma outra heurística de visão materializada hvm que reescreve a consulta Q para sugerir uma visão materializada v. Logo, o otimizador de consultas teria dois caminhos possíveis para gerar o plano de execução de Q:

i) utilizar o índice i para percorrer T e obter as tuplas necessárias para calcular a resposta de Q; ou

ii) ler a nova estrutura v com as respostas pré-processadas para responder Q.

Consequentemente, a decisão do otimizador vai optar por uma das soluções, e descartar a outra. Ou utilizará o índice i para percorrer a tabela

T, ou usará a VMA v e não precisará de acessar os dados de T. Neste cenário, i e v são soluções mutuamente exclusivas.

4.1.4 Solução inválida

Uma **solução inválida** é uma solução considerada inexequível devido a uma restrição tecnológica do sistema gerenciador de banco de dados.

O método de combinação, e os algoritmos apresentados aqui, não são específicos a um SGBD particular. São métodos genéricos e, teoricamente, podem ser aplicados a qualquer SGBD relacional. Já as instâncias das combinações que o método permite gerar são soluções sensíveis aos detalhes de implementação do SGBD usado.

Para que uma ferramenta automática possa gerar tais combinações, as restrições tecnológicas envolvidas precisam ser consideradas e tratadas pelo método de geração. É necessário reconhecer o que chama-se aqui de soluções inválidas.

Para ilustrar, seguem alguns exemplos de restrições:

(a) O PostgreSQL em sua versão 10 permite a criação das estruturas de acesso VMAs, INDs, PINs. Adicionalmente, é possível criar índices parciais e completos tanto em tabelas quanto em visões materializadas do banco de dados.

(b) O SQL Server 2017 possui uma versão limitada de índices parciais (chamada de *filtered index*) com restrições sobre quais tipos de atributos podem ser filtrados. Além disso, também não é possível criá-los em visões materializadas.

(c) O SGBD MySQL versão 8.0 não suporta índices parciais ou visões materializadas.

Assim a geração de soluções combinadas utilizando as técnicas de VMAs, índices completos e parciais não possui restrições tecnológicas no PostgreSQL 10; não pode ser realizada utilizando as três estruturas de acesso no MySQL 8.0; e utilizando o SQL Server 2017 é possível gerar soluções combinadas, mas não é possível gerar soluções com visões materializadas indexadas por índices completos ou parciais. Logo, há soluções corretas e teoricamente possíveis, mas não são válidas em um determinado SGBD.

4.1.5
Diferentes recursos usados como limites pelo método

Recursos computacionais não são infinitos durante uma tarefa de sintonia. Cada ajuste consome uma certa quantidade de recursos. Por exemplo, para combinar estruturas de acesso como visualizações materializadas, índices completos ou índices parciais, é necessário espaço em memória secundária para cada uma das estruturas adicionadas. Logo, o conjunto de ações que fazem parte da solução final precisa respeitar tais restrições enquanto tenta diminuir o custo de execução da carga de trabalho.

A abordagem de sintonia fina global permite que tais restrições sejam respeitadas para a atividade de sintonia fina como um todo, e não localmente para cada técnica avaliada. Quando se executa uma sintonia fina local para diferentes tipos de ações, e de forma independente, mesmo que cada ferramenta respeite as restrições isso não significa que a soma dos recursos consumidos permanecerá dentro dos limites. Afinal, uma ferramenta independente não considera os recursos consumidos por outras ferramentas executando em paralelo.

O método de seleção foi modelado para garantir que o conjunto final de ações não exceda os recursos computacionais disponíveis. Porém, o método é abrangente em relação ao tipo de recurso computacional será considerado. Três tipos de recursos possíveis são:

Espaço em memória secundária (disco): o DBA pode limitar o espaço de disco usado para armazenar as estruturas criadas no banco de dados;

Custo de execução: Cada solução possui um custo de execução predito pelo modelo de custo. Normalmente, se usa a unidade de medida particular de cada SGBD que é um somatório entre quantidade de páginas de disco e operações de CPU;

Tempo de execução: Caso o modelo de custo consiga predizer o tempo de execução das soluções, este recurso pode ser utilizado como uma restrição.

4.2
O método de combinação e seleção

Nesta tese propomos um método de combinação abrangente que segue o clássico ciclo automático [6, 3] de coletar informações sobre a carga de trabalho, gerar ações de sintonia fina, avaliá-las, e executá-las no banco de dados. As novas estratégias de *geração* e *seleção* de ações combinadas são as principais inovações em relação ao estado da arte. A Figura 4.1 apresenta os principais passos.

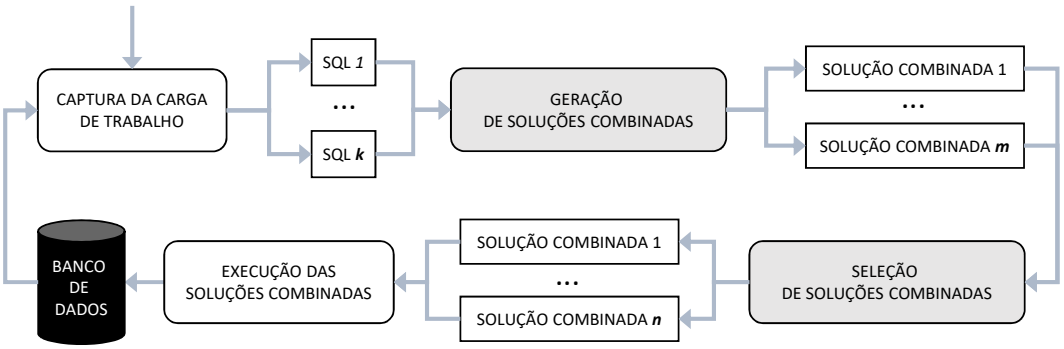


Figura 4.1: Etapas do método para combinação e seleção automática de ações de sintonia fina.

O método proposto aqui possui quatro etapas: i) captura da carga de trabalho; ii) geração de soluções combinadas; iii) seleção de soluções combinadas; e execução de soluções (Figura 4.1.

Captura da carga de trabalho: A carga de trabalho no contexto desta tese consiste nos dados armazenados no banco de dados e um conjunto de comandos SQL executados neste banco. O fluxo de execução do método (Figura 4.1), inicia-se com a captura da carga de trabalho, que seleciona dentre os comandos SQL executados no banco, k elementos representativos. Considera-se que a forma de escolha destes k comandos já foi tratada na literatura por outros trabalhos, e escolhemos por usar a proposta de Monteiro [6] sobre como selecionar tais comandos.

Geração de soluções combinadas: O passo de geração usa os k comandos selecionados na etapa de captura e gera m soluções combinadas. Note que o número de soluções combinadas m pode ser diferente do número de comandos SQL k, e o tamanho de m depende da quantidade de soluções combinadas enumeradas para cada comando SQL em k.

Seleção de soluções combinadas: No passo de seleção, as m soluções geradas são filtradas de acordo com as restrições computacionais. Como resultado, são selecionadas n soluções combinadas que diminuem o custo de execução dos k comandos SQL e cuja soma dos recursos computacionais necessários para a execução do conjunto n não exceda o limite dos recursos disponíveis. Logo, o tamanho de n varia de acordo com quantas soluções da entrada m precisam ser filtradas, então $n \leq m$.

Execução de soluções combinadas: O último passo é a execução das n soluções combinadas filtradas na fase de seleção. Esta execução deve ser feita em um momento oportuno da carga de trabalho, uma vez que pode interferir no desempenho e disponibilidade do banco para outros usuários. Existem diferentes estratégias para a execução de ações de sintonia fina mas,

de modo geral, são feitas em momentos de baixa demanda, como períodos noturnos ou janelas de manutenção estabelecidas pelo DBA.

A etapa de geração é descrita no Capítulo 5, e a de seleção no Capítulo 6. A Seção a seguir discute um conjunto de premissas desejáveis para o método automático.

4.3 Premissas para um método automático de combinação de ações de sintonia fina

Existem na literatura diferentes premissas para métodos automáticos de sintonia fina [3, 6, 26, 16]. Eles se diferenciam pela maneira como se acoplam ao banco de dados, pelas formas de capturar a carga de trabalho, pela execução paralela à carga de trabalho e outros. Assim como requisitos de software, a escolha de quais premissas seguir influenciam a modelagem do método e a estratégia usada para a geração das soluções combinadas. Logo, dado o contexto deste trabalho, são descritas a seguir as premissas desejáveis para o método automático de combinação, e suas justificativas.

4.3.1 Abordagem não-intrusiva

Existem ferramentas de sintonia fina que são incorporadas ao código-fonte do SGBD para gerar e avaliar soluções. Elas têm a vantagem de acessar o modelo de custo interno do otimizador para prever o impacto das ações na carga de trabalho. Na maioria dos casos, essa estratégia gera previsões mais eficientes do que ferramentas que usam modelos de custo independentes e externos ao SGBD. Por outro lado, a abordagem intrusiva depende da disponibilidade do código-fonte do SGBD, que muitas vezes é inacessível, já que muitos são softwares proprietários. Outra desvantagem é que, mesmo em um SGBD de código aberto (PostgreSQL por exemplo), as ferramentas de ajuste precisam ser revisadas e, geralmente, adaptadas a cada nova versão do software. Tudo isso torna a manutenção cara e altamente dependente da tecnologia e dos algoritmos internos do SGBD.

Considerando os pontos fortes e fracos da abordagem intrusiva, concluímos que: no contexto de uma método abrangente para combinação de ações, e de uma estrutura para reutilização em larga escala, um método não invasivo para a criação e avaliação de soluções de ajuste pode a) proporcionar mais oportunidades de reutilização b) diminuir o custo de manutenção da ferramenta, c) ser estendido para diferentes SGBDs através de interfaces que isolam as regras de combinação dos detalhes de implementação para cada um deles.

4.3.2 Execução offline

Ferramentas automáticas podem analisar cargas de trabalho e propor ações utilizando duas abordagens: online e *offline*. No método online a ferramenta executa um ciclo ininterrupto de monitoramento, análise e ajustes do banco de dados. Já o *offline* coleta informações durante uma janela de tempo pré-determinada, analisa os dados, gera ações, filtra as melhores, e executa-as no banco de dados. Um novo ciclo se inicia apenas com uma nova intervenção do DBA.

A principal diferença entre elas é o tempo disponível entre a análise da carga de trabalho e a execução das ações. Na estratégia online, a ferramenta precisa ter um tempo de resposta curto, e se adaptar às sazonalidades da carga de trabalho em tempo de execução. Já a *offline*, tem como prerrogativa que a carga de trabalho seja estável. Se a carga de trabalho mudar, uma nova tarefa de sintonia fina será realizada.

Vale ressaltar, que ao contrário da otimização do plano de execução realizada pelo otimizador do SGBD que precisa ser *online*, a tarefa de sintonia fina é por natureza *offline*. Alguns trabalhos da literatura [64, 65] ainda tratam a abordagem *online* como **contínua** (*continuous tuning*) para explicitar que é uma tarefa realizada *offline* mas continuamente para monitorar mudanças na carga de trabalho. Logo, não existe uma limitação crítica de tempo para a tarefa de sintonia fina como em aplicações *online* tradicionais. Encontrar um bom conjunto de ações de sintonia fina para a carga de trabalho pode consumir um tempo razoavelmente viável, de acordo com as expectativas do DBA.

Optou-se neste trabalho por adotar uma estratégia *offline*, justificada pela complexidade do processo de combinação. Dado que até então não existia um método automático de geração de soluções combinadas, muitos desafios emergiram durante a modelagem do método. Optar por uma abordagem *offline* permitiu usar algoritmos por força bruta neste primeiro momento de exploração e entendimento do problema.

O estudo e evolução para uma execução *online*, que consiga ajustar o banco de dados continuamente e seguindo as variações da carga de trabalho, é desejável. Mas dado o limite de escopo da tese, será registrada como um trabalho futuro. De qualquer forma, o modelo *offline* como proposto aqui permite o ajuste para estas mudanças, por uma iniciativa do DBA de iniciar um novo ciclo de ajustes.

4.3.3 Extensão para múltiplos SGBDs e modelos de custo

Cada SGBD possui diferentes algoritmos e estratégias de otimização de consultas customizadas implementados por seus engenheiros e cientistas de dados. Assim, cada SGBD possui particularidades e diferentes interfaces de acesso e protocolos de comunicação. Considera-se que a modelagem de um método para sintonia fina global seria uma ótima oportunidade para planejar uma única ferramenta para diferentes SGBDs.

Uma forma de estender ferramentas de sintonia fina para diferentes SGBDs é encapsular os detalhes de implementação em diferentes modelos de custo que possuam as mesmas interfaces. Isso permite que as particularidades sejam tratadas de modo abstrato pelo núcleo principal do software. A ideia é criar e avaliar ações de ajuste de uma maneira específica, com diferentes modelos de custo para cada SGBD, mas usando a mesma ferramenta de software.

Uma vantagem de suportar vários SGBDs é maximizar a reutilização do código-fonte e os custos de manutenção. Ferramentas instanciadas a partir desta abordagem podem propor e avaliar soluções para diferentes SGBDs com um único custo de desenvolvimento e evolução, além de permitir que o DBA administre uma única ferramenta.

Os modelos de custo encapsulados e independentes permitem a especialização de múltiplos modelos para um único SGBD. Isso abre a possibilidade para uso de diferentes modelos para diferentes cenários, como por exemplo:

(a) **diferentes tipos de carga de trabalho**: o DBA poderia alterar e estender os modelos de acordo com experiências pessoais criando diferentes versões para diferentes cargas.

(b) **diferentes plataformas**: pode-se ter modelos para servidores tradicionais e outros para ambientes virtualizados, onde sabemos que a camada de virtualização pode interferir nos custos de execução dos comandos SQL [66].

(c) **diferentes tecnologias de armazenamento**: Disco duro (HDD) e *Solid-State Drive* (SDD) possuem diferentes desempenhos para operações de leitura e escrita. Os SSDs são até 100 vezes mais rápidos para executarem leituras aleatórias em relação ao HDD, ao mesmo tempo que oferecem uma velocidade de gravação sequencial equivalente [67]. Isso significa que diferentes instâncias do mesmo SGBD precisarão de modelos de custo diferentes para prever os custos de operações em diferentes tecnologias de armazenamento.

4.3.4
Extensão para diferentes técnicas de sintonia fina

Existem propostas para muitos tipos de ações de sintonia fina na literatura: criar e manter estruturas de acesso como índices e visualizações materializadas, particionamento de banco de dados lógico e físico, reescrita de consulta, ajuste de parâmetro e outros. [3, 2, 68, 69].

Diferentes técnicas são necessárias para a combinação mas, ao mesmo tempo, não seria possível modelar todas as técnicas existentes no escopo deste trabalho. Então, definiu-se como requisito a capacidade do método de estender para a inclusão de novos tipos de ações de sintonia fina. Afinal de contas, é um método abrangente que, teoricamente, pode ser aplicado para outras técnicas além das já exploradas aqui.

4.3.5
Extensão para diferentes heurísticas de sintonia fina

Uma técnica (criação de índices completos, por exemplo) pode possuir diferentes heurísticas tanto para a geração, quanto seleção de soluções combinadas. Logo, decidiu-se permitir a inclusão de diferentes heurísticas nas duas etapas. A justificativa é a mesma de incluir diferentes técnicas: não seria possível modelar todas as existentes, e limitar o método a apenas aquelas estudadas aqui não condiz com a proposta de um método abrangente de combinação.

Um vantagem desta liberdade de extensão é a capacidade de comparação das heurísticas. É difícil comparar heurísticas implementadas em diferentes ferramentas e diferentes tecnologias. Uma vez que serão executadas na mesma arquitetura, em paralelo e avaliando as mesmas consultas, uma classificação de acordo com a eficácia e eficiência talvez seja mais justa. Esta comparação não é um dos objetivos deste estudo mas, seria algo possível pelas escolhas tomadas na modelagem do método de combinação proposto.

4.3.6
Resumo do capítulo

Neste Capítulo foram apresentadas as definições de soluções de sintonia adotados aqui. Discutiu-se as premissas para a combinação de ações de sintonia fina, assim como diferentes recursos usados como limites pelo método de combinação. No próximo Capítulo será descrito em detalhes o algoritmo proposto para a geração automática de soluções combinadas.

5
Geração automática de soluções combinadas de sintonia fina

Apresenta-se aqui um método automático e abrangente para geração de soluções combinadas para bancos de dados relacionais. Ao longo deste capítulo são listadas as escolhas que levaram à definição dos algoritmos para o método de geração e suas justificativas. A maioria das escolhas foram influenciadas pelos requisitos (Seção 4.3) mas dois deles, em particular, são importantes para o algoritmo de geração: a capacidade de extensão para diferentes técnicas, e diferentes algoritmos. Estes requisitos exigem que o método de geração de soluções combinadas seja abrangente o suficiente para a inclusão de técnicas e heurísticas já existentes. Para cumprir este requisito, propomos duas definições chamadas de especialista em sintonia fina e comitê de especialistas.

5.1
Especialista

Existem na literatura diferentes técnicas de sintonia fina e, para cada técnica, múltiplos métodos de geração para cada tipo de ação. É possível reescrever uma consulta para gerar visões materializadas usando diferentes métodos [20, 70, 71, 72, 31, 27, 73], assim como existem diversas formas de se propor índices completos [74, 75, 76, 77, 78].

Um especialista em sintonia é uma abstração de um algoritmo único de geração de soluções combinadas a partir de uma técnica específica. Por exemplo, um algoritmo capaz de reescrever uma consulta SQL e gerar uma ou mais visões materializadas. Ou ainda um algoritmo que, a partir de algum método particular, analisa a carga de trabalho e propõe um conjunto de índices candidatos. Podem existir dois ou mais especialistas de uma mesma técnica (VMAs, por exemplo), desde que usem algoritmos diferentes.

5.2
Comitê de especialistas

Um comitê de especialistas é o nome dado a um grupo de especialistas acordados a gerar soluções combinadas durante uma tarefa de sintonia fina em um determinado banco de dados.

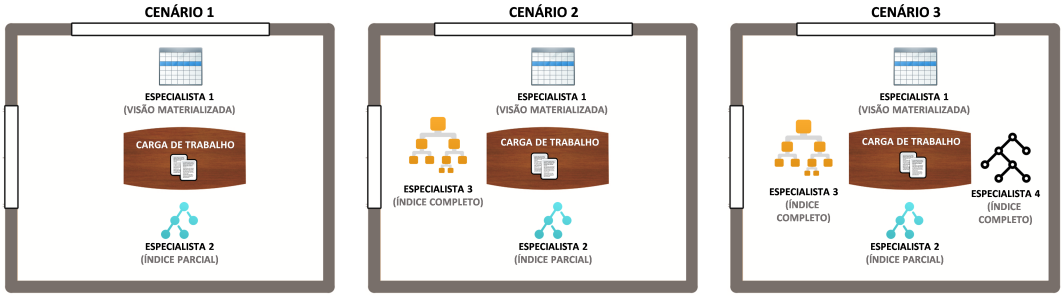


Figura 5.1: Possíveis cenários de comitês de especialistas em sintonia fina seguindo as premissas de formação.

A inspiração para esta definição veio do comportamento natural de um possível grupo de especialistas humanos em sintonia fina, DBAs por exemplo. Tais técnicas são naturalmente complexas e envolvem muitas variáveis a serem consideradas. Um DBA normalmente é especializado em um subconjunto delas, e quando trabalha em grupo com outros DBAs com especialidades diferentes, precisa colaborar, trocar e negociar com os outros indivíduos do grupo para que as soluções propostas não sejam somente planejadas localmente, mas também considerem as outras soluções propostas pelos demais elementos do grupo através de uma análise global do problema.

Propomos as seguintes premissas para a formação deste comitê:

(i) O comitê deve ter dois ou mais especialistas, para que seja possível a combinação.

(ii) Teoricamente não existe um número máximo de especialistas, mas existe um limite prático em relação aos custos computacionais para executar a tarefa.

(iii) Não existe um conjunto de especialistas fixo. Novos especialistas podem ser adicionados sem a necessidade de modificações no método de combinação.

(iv) Não existe ordem de execução para os especialistas.

(v) Não pode existir duas instâncias do mesmo especialista.

A Figura 5.1 ilustra possíveis cenários para a combinação de comitês de especialistas que respeitam as premissas propostas. Cenário 1: dois especialistas (número mínimo) se juntam em uma "sala de reunião virtual"para planejar soluções de sintonia fina global. Os dois têm especialidades diferentes: um em visões materializadas e outro de índices parciais. Cenário 2: existem três especialistas, também um de cada especialidade (IND, PIN, VMA), e exemplifica que não há um número fixo de especialistas. Por fim, Cenário 3:

ilustra um caso onde existem múltiplos especialistas, e mais de um especialista do mesmo tipo de ação (índice completo) onde cada elemento representa um algoritmo diferente de geração (Especialista 3 e Especialista 4).

Um questionamento válido à proposta de um comitê de especialistas é: Por que incorrer na complexidade de modelar especialistas ao invés de tratá-los apenas como algoritmos de geração e seleção?

Para tentar respondê-la é importante salientar a complexidade de gerar uma ação de sintonia fina (tanto local quanto global). Técnicas para geração e seleção automática de um único tipo de ação usando um único algoritmo são objetos de estudo de um número considerável de dissertações [79, 80, 69, 52, 81, 14, 82], e teses [6, 83, 23]. Vale ressaltar inclusive que encontrar a solução ótima para algumas técnicas é, na prática, inviável. Por exemplo, selecionar um conjunto apropriado de VMAs que minimize o tempo total de resposta da carga de trabalho submetida ao SGBD é um problema NP-Difícil [20] assim como o problema de seleção de índices [44].

Respondendo ao questionamento: a abordagem de geração proposta precisa combinar múltiplas ações que, sozinhas, já são complexas. O comitê de especialistas é uma abstração que quebra o problema em partes. Dividir e conquistar. Propomos aqui não apenas um método teórico de como realizar a combinação, mas também uma estratégia passível de implementação e viável na prática. Entende-se que a implementação destes algoritmos complexos em um bloco único de software seria muito difícil, se não inviável.

O Capítulo 7 apresenta uma implementação com resultados práticos em três SGBDs populares do mercado. Considera-se que as unidades de software independentes e autônomas (especialistas) organizadas em um comitê de sintonia foram o ponto chave para automatizar o processo de combinação, por dividir o conhecimento (e complexidade) em diferentes nós de processamento, e tratar as exceções (soluções inválidas, soluções ineficazes, soluções competitivas, etc) apenas no escopo de cada especialista e sua respectiva técnica de sintonia.

5.3 O problema de controle

A abordagem proposta tenta realizar a combinação de diferentes tipos de ações através da representação de especialistas como partes autônomas de um sistema inteligente de software. Porém a estratégia de dividir o problema em partes independentes sofre do clássico "problema de controle" (*Control Problem*) estudado pela área de Inteligência Artificial (IA) [84].

Na tentativa de resolver um problema de certo domínio, um sistema com

unidades autônomas executa uma série de ações para chegar à solução. Cada ação é disparada por dados ou soluções geradas anteriormente, e aplica alguma fonte de conhecimento do domínio para gerar ou modificar a solução corrente. Em cada ponto do processo de solução, várias dessas ações podem ser possíveis. Logo, o problema de controle é: *dentre as potenciais ações, quais um sistema com unidades autônomas deve executar em cada ponto do processo de solução do problema?* [84].

O problema do controle é fundamental para todos os processos cognitivos e sistemas inteligentes. Ao resolvê-lo, um sistema decide, de forma implícita ou explícita, quais problemas ele tentará resolver, que conhecimentos ele trará e quais métodos e estratégias de solução de problemas serão aplicados. Também decide como avaliará soluções de problemas alternativos, como saberá quando problemas específicos serão resolvidos e sob quais circunstâncias interromperá sua execução para problemas ou subproblemas selecionados. Assim, na resolução do problema de controle, um sistema determina seu próprio comportamento cognitivo.

Existem três questões principais para resolver o problema de controle que foram adaptadas para o problema específico de geração de soluções de sintonia fina global:

1. Como combinar e compor as soluções?
2. Como coordenar múltiplas fontes de conhecimento (especialistas) na busca pela solução?
3. Quais circunstâncias serão condições de parada?

Apresenta-se a seguir uma solução para cada uma das três questões.

5.3.1 Como combinar e propor as soluções

Propõe-se uma estratégia de força bruta para combinar ações, mas aplicando restrições ao espaço de busca devido à complexidade de se testar todas as alternativas. O objetivo é desenvolver as soluções válidas e descartar aquelas inválidas. Considerando que nem todas as soluções precisam ser testadas (vide Capítulo 4), estuda-se aqui a aplicação da meta-heurística de otimização *branch-and-bound* (BnB) [10] para a geração das soluções.

O método *branch-and-bound* baseia-se na ideia de desenvolver uma enumeração inteligente das soluções combinadas à solução ótima de um problema. Este método foi escolhido por dois motivos: i) permite a modelagem de regras de poda (*bound*) utilizando regras do domínio do problema, conveniente para

o problema de sintonia fina, e ii) permite uma avaliação de cada estado por unidades independentes.

No contexto da geração de soluções combinadas, o *BnB* analisa a cada estado a possível combinação entre dois especialistas diferentes, e cada especialista conhece as restrições e limitações da técnica que é especialista em relação à técnica que propõe a combinação. Durante a modelagem do BnB foram tomadas as seguintes decisões em relação aos componentes básicos do algoritmo:

- **Regra de seleção**: busca em largura. A técnica recursivamente avalia a cada iteração um subconjunto de estados utilizando uma estratégia de busca em largura. Foi escolhida para permitir uma avaliação assíncrona de cada estado pelos especialistas dentro de cada iteração.
- **Regra de expansão**: Se o estado corrente gerar uma solução válida, então ele será expandido.
- **Regra de poda dura (ou tradicional)**: São propostas duas regras de eliminação:
  1. se o estado corrente gerar uma solução inválida, então ele não será expandido. Considera-se que todas as soluções combinadas geradas a partir de um estado inválido também serão soluções inválidas.
  2. se o estado corrente gerar um custo para executar o comando SQL maior ou igual ao estado anterior, então ele não será expandido.
- **Regra de poda relaxada**: São propostas duas regras de eliminação:
  1. se o estado corrente gerar uma solução inválida, então ele não será expandido;
  2. se o estado corrente gerar um custo para executar o comando SQL maior ou igual ao custo original do comando SQL, então ele não será expandido.
- **Regra de divisão**: O estado corrente gera todos os possíveis estados a partir do subconjunto de especialistas que ainda não contribuíram em estados anteriores daquela solução corrente.
- **Condição de parada**: A execução é terminada quando foram feitas todas as tentativas de expansão a partir dos estados ativos.

Dentre as regras definidas para a versão proposta do BnB, realizou-se um estudo adicional da regra de eliminação onde propôs-se duas regras de poda: tradicional e a relaxada. Ambas podam quando o estado corrente gera uma

solução inválida, mas a tradicional elimina as soluções que geram um menor benefício que o estado anterior, enquanto a versão relaxada elimina apenas aquelas que o estado corrente gera um custo para executar o comando SQL maior ou igual ao custo original do comando – ou seja – qualquer solução que diminua o custo original de executar o comando SQL é considerado um estado ativo e será expandido.

O relaxamento da regra de poda aumenta o espaço de busca em relação à regra "dura", mas foi investigada como parte da avaliação do algoritmo de geração. O método aqui proposto possui duas fases claramente distintas: geração e seleção. Na fase de geração decidiu-se por propor uma alternativa onde são avaliadas também todas as soluções consideradas válidas, mesmo que elas (naquele momento) não sejam boas alternativas. Ambas as versões são discutida em detalhes na Seção 5.4.

5.3.2 Coordenando múltiplas fontes de conhecimento (especialistas) na busca pela solução

Uma vez definido como gerar as soluções combinadas, trata-se aqui de como coordenar múltiplos especialistas durante as iterações do algoritmo BnB. A solução proposta é o uso de um *blackboard*, um *Framework* proposto por Erman et al [85], inicialmente para a tarefa de reconhecimento de fala, e que desde então vem sendo usado em diversos outros domínios. O *blackboard* trata a resolução de problemas como um processo incremental e oportuno de montar a solução final através de um conjunto de soluções satisfatório [84].

O *blackboard* é usado aqui como um repositório compartilhado. Cada especialista lê o estado corrente no *blackboard*, tenta propor uma nova solução combinada, e escreve sua nova solução no *blackboard* novamente. Todos os especialistas trabalham juntos simultaneamente para resolver o problema realizando uma ou ambas intervenções a seguir: (a) modificam ou estendem solução(ões) dada(s) anteriormente por outra fonte de conhecimento; (b) adicionam sua própria, e inédita, solução para resolver o problema.

Uma das vantagens dessa arquitetura baseada no *blackboard* é escalabilidade e dinamicidade. Especialistas podem ser facilmente adicionados ou removidos do método de geração sem necessidade de um controle centralizado. Os especialistas são independentes entre si e, portanto, podem funcionar concorrente e assincronamente, inclusive em diferentes nós de processamento.

Hayes-Roth [84] propõe três suposições básicas ao utilizar o *blackboard* que foram estendidas e adotadas aqui:

1. Todos os elementos da solução, gerados durante a combinação, são

registrados em um banco de dados compartilhado e estruturado chamado *blackboard*.

2. Elementos de solução são gerados e registrados no *blackboard* por processos independentes chamados de especialistas.

3. Em cada ciclo de solução de problemas, um mecanismo de programação escolhe uma única tarefa para executar e propor uma solução (iteração do algoritmo *Branch-and-Bound*).

5.3.3 Condições de parada

A execução é terminada quando foram feitas todas as tentativas de combinação válidas.

Uma vez que os três problemas de alto nível do problema de controle foram respondidas, temos então uma visão abstrata do algoritmo de combinação. Descreve-se a seguir o algoritmo de geração de soluções combinadas em detalhes, seguido de um exemplo de execução.

5.4 Algoritmo de geração de soluções combinadas

Propomos duas regras de poda para o algoritmo *branch-and-bound*, uma tradicional que expande apenas o melhor estado de cada iteração, e uma relaxada que expande todos os estados que diminuem o custo de execução original do comando SQL avaliado. Esta proposta da poda relaxada surgiu como uma alternativa a ser explorada durante discussões sobre a redução do espaço de busca da poda "dura". Durante a investigação destes limites, algumas hipóteses foram levantadas, e que justificam o estudo de uma versão relaxada da regra além da poda "dura":

a) no momento da poda, talvez a solução avaliada ainda não seja uma boa solução somente por uma tendência sazonal da carga de trabalho, ou por uma deficiência do modelo de custo usado para calcular o ganho potencial, já que este ganho é uma predição e está sujeito a ruídos;

b) a fase de seleção pode ser executada múltiplas vezes utilizando-se diferentes heurísticas de seleção, e talvez soluções com menor benefício na fase de geração podem ter um desempenho diferente quando avaliadas globalmente por diferentes modelos de custo;

Uma vez definidos todos os componentes básicos de instanciação do algoritmo BnB, são apresentados aqui o algoritmo de Geração de Soluções

Combinadas (GSC) (Algoritmo 1) e sua versão relaxada Algoritmo de Geração de Soluções Combinadas Relaxado (GSCR) (Algoritmo 2). O algoritmo GSC evita que soluções que não caminham para a solução ótima sejam expandidas e, consequentemente, uma maior redução no espaço de busca. Já o GSCR

implementa a regra de poda relaxada, e explora um espaço de busca maior.

Algoritmo 1: Algoritmo de Geração de Soluções Combinadas (GSC)

```
   Entrada: sql, especialistas, modelo_custo.
   Saída: solucoes_selecionadas
1  início
2      Q ← {sql};
3      solucoes_selecionadas ← ∅;
4      melhor_C ← {sql};
5      menor_custo ← ∞;
6      para P ∈ Q faça
7          custo_corrente ← custo executar sql usando P calculado
            pelo modelo_custo;
8          se custo_corrente < menor_custo então
9              menor_custo ← custo_corrente;
10             S ← todos os estados expandidos pelos especialistas a
                partir de P calculado pelo modelo_custo;
11             se S é uma solução válida então
12                 para C ∈ S faça
13                     custo_corrente ← custo executar sql usando C
                        calculado pelo modelo_custo;
14                     se custo_corrente < menor_custo então
15                         menor_custo ← custo_corrente;
16                         melhor_C ← C;
17                     fim
18                 fim
19             fim
20             se melhor_C ∉ solucoes_selecionadas então
21                 solucoes_selecionadas ←
                    solucoes_selecionadas ∪ {melhor_C};
22             fim
23             se melhor_C ∉ Q então
24                 Q ← Q ∪ {melhor_C};
25             fim
26         fim
27     fim
28     retorna solucoes_selecionadas
29 fim
```

Os algoritmos GSC e GSCR possuem os mesmos dados de entrada:

Algoritmo 2: Algoritmo de Geração de Soluções Combinadas Relaxado (GSCR)

```
Entrada: sql, especialistas, modelo_custo.
Saída: solucoes_selecionadas
1  início
2      Q ← {sql};
3      solucoes_selecionadas ← ∅;
4      menor_custo ← ∞;
5      melhor_C ← {sql};
6      custo_original ← custo original de executar sql sem nenhuma
         ação calculado pelo modelo_custo;
7      para P ∈ Q faça
8          custo_corrente ← custo executar sql usando P calculado
             pelo modelo_custo;
9          se custo_corrente < custo_original então
10             se custo_corrente < menor_custo então
11                 menor_custo ← custo_corrente;
12             fim
13             S ← todos os estados expandidos pelos especialistas a
                 partir de P calculado pelo modelo_custo;
14             se S é uma solução válida então
15                 se S não é uma solução mutuamente exclusiva então
16                     para C ∈ S faça
17                         custo_corrente ← custo executar sql usando C
                             calculado pelo modelo_custo;
18                         se custo_corrente < menor_custo então
19                             menor_custo ← custo_corrente;
20                             melhor_C ← C;
21                         fim
22                         se custo_corrente < custo_original então
23                             Q ← Q ∪ {melhor_C};
24                         fim
25                     fim
26                 fim
27             fim
28             se melhor_C ∉ solucoes_selecionadas então
29                 solucoes_selecionadas ←
                     solucoes_selecionadas ∪ {melhor_C};
30             fim
31         fim
32     fim
33     retorna solucoes_selecionadas
34 fim
```

– **sql**: um comando SQL capturado da carga de trabalho ao qual se deseja diminuir o custo de execução;

– **especialistas**:uma lista de métodos independentes para gerar soluções combinadas capazes de diminuir o custo de execução do comando SQL;

– **modelo_custo**: para avaliar as soluções combinadas, gerando predições como custo de execução e benefícios. Cada SGBD pode possuir um modelo de custo específico.

Os dois algoritmos também possuem a mesma saída:

– **solucoes_selecionadas**: o conjunto de soluções combinadas que minimizam o custo de execução do comando *sql* dado como entrada.

A principal diferença entre o GSC e a versão relaxada GSCR pode ser observada nas comparações da linha 8 do GSC e linha 9 do GSCR. O GSC compara o custo de executar o comando *sql* na presença da solução P é menor que o valor armazenado na variável *menor_custo*, que é o menor custo encontrado durante toda a execução. Isso faz com que ao final da expansão de todas as soluções do estado corrente, a variável *melhor_P* armazenará a melhor solução, e apenas ela será adicionada à lista de estados ativos (linha 24) para serem expandidos na próxima iteração. Já na versão relaxada GSCR a verificação da linha 9 compara o custo de executar o comando *sql* na presença da solução P apenas com o custo original do comando *sql*, ou seja, sem nenhuma ação de sintonia fina. A versão GSCR garante apenas que a solução P não piore o desempenho do comando original, e que qualquer solução que diminua o custo de execução do comando *sql* seja considerado um estado ativo (linha 22), e portanto seja expandido. A seção a seguir apresenta um exemplo de execução do Algoritmo GSC para ilustrar a proposta.

5.5 Eliminação de soluções mutuamente exclusivas

A eliminação de soluções mutuamente exclusivas pode reduzir o espaço de busca pelas melhores soluções combinadas, e economizar recursos computacionais. O algoritmo GSC evita a geração de tais soluções pela aplicação da regra de poda dura. A cada iteração apenas o caminho de menor custo é escolhido como estado ativo para ser expandido na próxima iteração. Isso faz com que os estados que gerariam soluções mutuamente exclusivas sejam naturalmente eliminados.

Por outro lado, a versão relaxada GSCR permite que mais de um estado seja escolhido para expansão a cada iteração. Logo, possibilita a geração de soluções mutuamente exclusivas. Para resolver este ponto, adicionou-se uma verificação extra ao Algoritmo GSCR (linha 16, Algoritmo 2) onde cada solução daquela iteração é testada pelos especialistas para identificação de tal condição.

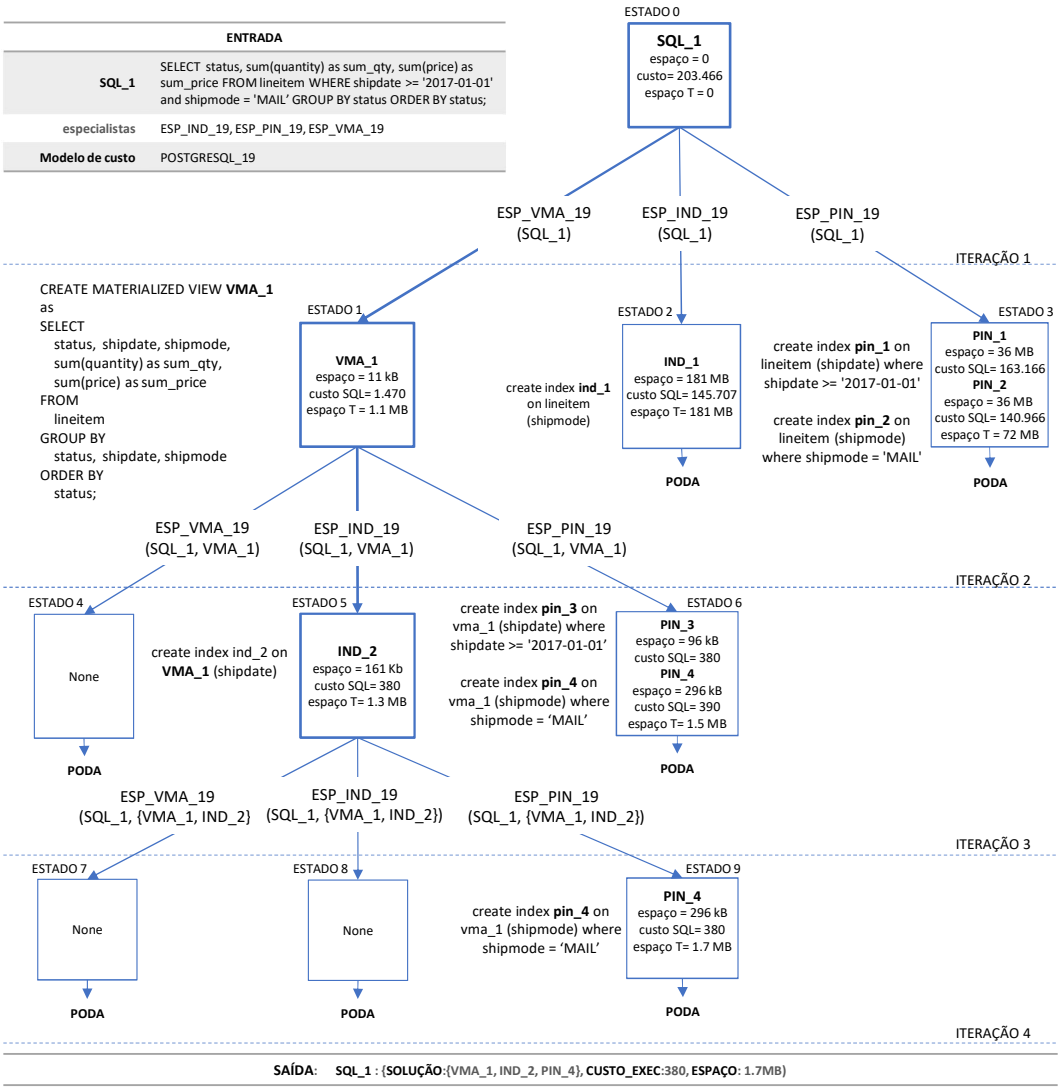


Figura 5.2: Possíveis cenários de comitês de especialistas em sintonia fina

5.6 Exemplo de execução do Algoritmo GSC

Como discutido anteriormente, combinações podem ser relativamente complexas de se gerar. A Figura 5.2 mostra uma execução possível do algoritmo GSC para tentar, com um exemplo simples mas realista, descrever o passo a passo do algoritmo. As iterações são detalhadas em cada passo com valores reais de uma execução extraída durante os experimentos utilizando o SGBD PostgreSQL.

No exemplo da Figura 5.2, tem-se como entrada do algoritmo:

(a) **SQL_1**: um comando SQL do tipo consulta que sumariza a quantidade de itens vendidos *(sum(quantity) as sum_qty)* e o valor *(sum(price)* as *sum_price*) da tabela *lineitem* que tenham sido despachados na data igual ou superior a *'01/01/2017'* , *(shipdate >= '2017-01-01')*, modo de envio

correios *(shipmode = 'MAIL')*, agrupados e ordenados pela situação do item *(GROUP BY status ORDER BY status)*;

(b) **ESP_IND_19, ESP_PIN_19, ESP_VMA_19**: três especialistas: um especialista em índices completos estendido do algoritmo de Monteiro [6], outro especializado em índices parciais a partir do algoritmo de Dominguez [52], e outro estendido a partir do algoritmo de geração de visões materializadas estendido a partir do algoritmo de Agrawal [20], respectivamente.

(c) **POSTGRESQL_19**: modelo para predição de custos de estruturas de acesso relativos à criação (processamento para geração da estrutura e persistência em memória secundária) e benefício da estrutura para a carga de trabalho. Foi criado a partir das propostas de [6], [26] e [52], que separadamente predizem custos para índices, visões materializadas e índices parciais. Este trabalho unificou estes modelos e estendeu-os para contemplar visões materializadas indexas por índices completos e índices parciais no SGBD PostgreSQL.

5.6.1 Primeira iteração

O comando SQL gera o estado *ESTADO 0* que possui um custo de execução de 203.466 em unidades de custo [86] do PostgreSQL e os recursos computacionais considerados pelo modelo de custo, neste caso, espaço em memória secundária. Note que o *ESTADO 0* representa a execução do comando sem nenhuma ação de sintonia fina, um estado que também pode ser selecionado como uma solução final caso nenhum outro estado diminua o seu custo de execução.

5.6.2 Segunda iteração

O algoritmo aplica a regra de expansão do BnB. Cada um dos especialistas listados como entrada gera soluções combinadas a partir do estado ativo *ESTADO 0*. Isso resulta na criação do ESTADO 1 pelo especialista *ESP_VMA_19* que propõe a ação *VMA_1* ; *ESTADO 2* pelo especialista *ESP_IND_19* com a ação *IND_1* ; e *ESTADO 3* pelo especialista *ESP_PIN_19* que sugere duas ações: *PIN_1* e *PIN_2* .

A segunda iteração mostra o primeiro passo da combinação, onde cada especialista cria uma sub-árvore com sua solução como raiz da árvore a ser expandida nas iterações seguintes. Neste exemplo, os estados *ESTADO 2* e

ESTADO 3 não são selecionados como estados ativos, já que a regra de poda do algoritmo GSC seleciona apenas a melhor solução a cada iteração, neste caso *ESTADO 1*. Vale ressaltar que, caso aplicássemos a regra de poda relaxada do algoritmo GSCR, ambos *ESTADO 2* e *ESTADO 3* seriam expandidos gerando um número maior de estados mas, consequentemente, testando um maior número de possíveis combinações.

5.6.3 Terceira iteração

O único estado ativo *ESTADO 1* é expandido para os estados *ESTADO 4* , *ESTADO 5* e *ESTADO 6* . O estado *ESTADO 4* não possui nenhuma ação de sintonia fina, já que o especialista *ESP_VMA_19* propôs a solução do *ESTADO 1* na iteração anterior. O *ESP_IND_19* propõe o *ESTADO 5* com a criação de um índice *IND_2* sobre a visão materializada *VMA_1* , e o *ESP_PIN_19* propõe o *ESTADO 6* com a criação dos índices parciais *PIN_3* e *PIN_4* sobre a *VMA_1* .

Nesta iteração o *ESTADO 5* é selecionado como estado ativo por apresentar o menor custo. Note que o *ESTADO 6* mostra uma solução com dois índices parciais *PIN_3* e *PIN_4* para a *VMA_1* onde o menor custo da solução (*PIN_3*) tem o mesmo custo predito do estado *ESTADO 5* (*IND_2*), porém o *ESTADO 5* foi executado primeiro pela regra de seleção de busca em largura. Logo, o *ESTADO 5* continua sendo o estado ativo da iteração.

O fato dos custos e benefícios serem predições, motivaram o estudo da versão relaxada da regra de poda e do algoritmo GSCR. Um índice completo pode ser mais genérico que um índice parcial e beneficiar um número maior de consultas. Entretanto, caso o comando avaliado possua uma frequência alta na carga de trabalho, teria-se deixado de expandir uma boa solução como o *ESTADO 6* e todas as suas variações posteriores. Em alguns casos onde a diferença entre as soluções é muito pequena, a regra de poda do algoritmo GSC pode ser restritiva e não expandir estados que possivelmente trariam benefícios maiores para a consulta avaliada. O Capítulo 8 explora melhor esta discussão.

5.6.4 Quarta iteração

O especialista *ESP_VMA_19* tenta expandir para o *ESTADO 7* mas não consegue propor nenhuma ação, uma vez que já contribuiu para esta solução combinada no *ESTADO 1*. O mesmo ocorre para o especialista *ESP_IND_19* que contribuiu para a solução no *ESTADO 5* . O *ESP_PIN_19*, único que

ainda não contribuiu nesta linha de execução, sugere então o índice parcial *PIN_4* .

Na terceira iteração o especialista *ESP_PIN_19* sugeriu o *PIN_3* e *PIN_4* para a *VMA_1*, mas na iteração 4 a coluna shipdate já possui uma proposta de índice (*IND_2*). Logo, *PIN_3* seria uma solução inválida para o estado atual da combinação, então apenas o *PIN_4* é sugerido.

5.6.5 Parada

Após a expansão por todos os especialistas na iteração 4, nenhum estado é selecionado como estado ativo, já que a solução proposta no *ESTADO 6* não diminuiu o custo de execução do comando SQL analisado. Logo, a execução é finalizada pela condição de parada em que todos as expansões possíveis foram testadas e a fila de estados ativos está vazia.

5.7 Resumo do capítulo

Este capítulo propôs as abstrações de especialista de sintonia fina como uma unidade de software autônoma e cognitiva, o comitê de especialistas, e o problema de controle decorrente desta abordagem. Para cada uma das três perguntas principais para solução do problema de controle foram apresentadas respostas (Seções 5.3.1, 5.3.2 e 5.3.3). Definiu-se o algoritmo de geração GSC capaz de gerar soluções combinadas de sintonia fina, sua versão relaxada GSCR, suas principais funções e decisões de implementação. Por fim, detalhou-se uma execução do GSC para ilustrar passo-a-passo com valores extraídos de um cenário real de teste.

6
Seleção automática de soluções combinadas

Dado um conjunto de ações de sintonia fina, escolher um subconjunto das melhores ações que satisfaçam a algum limite pré-determinado é uma tarefa difícil, e normalmente envolve testes de diferentes permutações entre as ações. Este Capítulo apresenta a fase de seleção do método de combinação proposto. O algoritmo proposto filtra o conjunto final de soluções combinadas que minimizam o custo global de execução da carga de trabalho, e respeitam os recursos computacionais. A Seção 6.1 apresenta o algoritmo de seleção, e a Seção 6.2 apresenta o modelo de custo proposto para esta fase de seleção.

6.1
Algoritmo de seleção de ações combinadas

Para o problema de seleção escolheu-se o algoritmo de otimização combinatória da mochila [87], mais precisamente, o problema da mochila booleana. Modelou-se o algoritmo para preencher a mochila com o maior valor possível (benefício) e respeitando o peso máximo da mochila - neste caso os recursos computacionais disponíveis para a tarefa de sintonia.

O problema da mochila possui um algoritmo guloso com complexidade $O(2^n)$ e é NP-Hard [87]. Considerando que o número de soluções combinadas é equivalente ao número de comandos SQL analisados, este número pode crescer de acordo com a janela de tempo de monitoramento do banco. Logo, decidimos por utilizar um algoritmo baseada em programação dinâmica [88], conhecida como "recursão com apoio de uma tabela". Ela possui complexidade $O(nc)$, onde n é o número de soluções combinadas que podem ser adicionadas na mochila (soluções combinadas), e c é a capacidade da mochila, ou seja, a quantidade de recursos computacionais disponíveis para a execução das soluções (espaço em disco, por exemplo).

A escolha deste algoritmo baseado em programação dinâmica se deu por dois motivos: i) possui complexidade menor que a versão gulosa, e ii) implementação recursiva relativamente simples quando comparado com soluções mais sofisticadas, como algoritmo da mochila utilizando o *Branch-and-Bound* [89]. Porém esta abordagem possui uma limitação: os pesos dos objetos e o limite da mochila devem ser inteiros, não negativos. Considerando o problema de

seleção de soluções combinadas, o algoritmo proposto contorna esta limitação.

Na modelagem proposta, o peso de uma solução é a quantidade de recursos computacionais que ela consome, logo, pode ser a) espaço em memória principal ou secundária, b) custo de CPU, ou c) tempo para execução das soluções. No caso de espaço em disco, o SGBD normalmente utiliza a medida de páginas de disco ocupadas, um número que será inteiro maior ou igual a zero. Em caso de CPU, o modelo de custo (e até planos de execuções dos SGBDs) utilizam inteiros positivos para descrever o número de operações de processamento. E no caso de ambos tempo de execução ou espaço em memória principal, pode-se converter números reais (1,5 horas / 1,5GB) em números inteiros (90 minutos / 1536MB).

Apesar do algoritmo de programação dinâmica ser mais eficiente que a versão gulosa, mesmo $O(nc)$ pode se tornar inviável caso c se torne um número muito grande. Adotou-se aqui a solução proposta por Monteiro [6] que utiliza uma diferente versão do algoritmo de programação dinâmica da mochila para selecionar índices completos, e propõe uma solução simples, mais eficiente, para diminuir a influência de c no tempo de execução: ao invés de utilizar a unidade de peso padrão da mochila, utilizar como unidade o tamanho do menor objeto avaliado. Logo, estratégias como esta que reduzem o tamanho de c, e consequentemente, podem reduzir substancialmente a complexidade do algoritmo.

O Algoritmo 3 descreve o método proposto chamado Seleção de Soluções Combinadas (SSC). Possui os seguintes parâmetros de entrada:

- c: (Capacidade da mochila) - recursos computacionais disponíveis;
- n: número de soluções combinadas a serem avaliadas;
- W: (pesos) vetor com os custos de execução em relação ao recurso computacional de cada uma das soluções combinadas;
- V: (valores) vetor com os benefícios hipotéticos necessários para a execução de cada uma das soluções combinadas;
- $suporte[n][c]$: matriz inicializada com valores vazios $\emptyset$;

e parâmetros de saída:

- $beneficio$: Benefício total de todas as soluções combinadas selecionadas;
- M: vetor de soluções combinadas selecionadas;

O procedimento recursivo SSC se inicia com o teste do valor armazenado em $suporte[n][c]$ (linha 4): Se existir qualquer valor que não o instanciado (vazio), significa que o algoritmo já testou a inclusão da solução combinada

Algoritmo 3: Algoritmo para Seleção de Soluções Combinadas (SSC)

```
   Entrada: c,n,W,V,suporte[n][c]
   Saída: beneficio, M
 1 início
 2   Procedimento SSC(n, c, M)
 3   início
 4     se suporte[n][c] ≠ ∅ então
 5       retorna suporte[n][c]
 6     fim
 7     se (n = 0) ou (c = 0) então
 8       resultado ← (0, ∅);
 9     senão
10       se w[n] > c então
11         resultado ← SSC(n − 1, c);
12       senão
13         (beneficio_com, M_com) ←
             V[n] + SSC(n − 1, c − W[n], M);
14         (beneficio_sem, M_sem) ← SSC(n − 1, c, M);
15         se beneficio_com > beneficio_sem então
16           resultado ← (beneficio_com, {M_com ∪ n});
17         senão
18           resultado ← (beneficio_sem, M_sem);
19         fim
20       fim
21     fim
22     suporte[n][c] ← resultado;
23     retorna resultado
24   fim
25 fim
```

na posição n e, portanto, pode apenas retornar o valor armazenado (linha 5). Caso a solução não tenha sido testada, o algoritmo prossegue e testa se todas as soluções já foram testadas ($n = 0$) ou se todo o espaço da mochila foi utilizado ($c = 0$) (linha 7). Se sim para qualquer um dos testes, o resultado para aquele estado é registrado como $(0, \emptyset)$, o que significa que nenhum recurso foi consumido, e nenhuma solução selecionada.

Na segunda parte do Algoritmo 3 (linhas 9 a 21), calcula-se para o estado corrente o benefício hipotético de colocar a solução combinada n na mochila (linha 13) e a solução final sem aquela solução (linha 14). Então ele compara os dois valores (linha 15) e decide se a solução fará parte ou não do conjunto final (linhas 16 e 17). Note que a variável *resultado* armazena uma tupla contendo o benefício corrente e um vetor M, que armazena a solução da posição n caso aquela solução seja colocada dentro da mochila (linha 16). Por fim, o algoritmo

armazena na posição $suporte[n][c]$ o valor da variável $resultado$ (linha 22) para ser usado em futuras iterações.

6.2 Modelo de Custo

Um modelo de custo para ações de sintonia fina considera, entre outros, dois fatores principais: o tipo da ação (VMA, IND, PIN, outros), e o SGBD usado. Para cada técnica, existem na literatura diferentes modelos de custos propostos para predizer o custo e o benefício das ações daquele tipo. E para cada SGBD, estes modelos específicos são adaptados para as especificidades de cada um.

Na seleção de ações independentes, uma estratégia popular é calcular o benefício e custo de cada ação, ordená-las pelo benefício e selecionar as top-k melhores. Já a estratégia proposta aqui seleciona através do algoritmo da mochila qual o melhor subconjunto de soluções combinadas de acordo com as restrições de recursos.

Observe que uma solução combinada envolve duas ou mais ações geradas para cada comando da carga de trabalho. Porém, como ilustrado na Figura 6.1, duas soluções combinadas podem compartilhar uma mesma ação.

Dado que o custo e benefício das ações independentes são calculados da mesma forma que nos trabalhos da literatura, descrevemos aqui como fazê-lo no contexto de soluções combinadas. Para ações independentes, os trabalhos de Shasha [2], Bruno [3], Monteiro [6] e Araújo [69] são alguns dos que discutem em detalhes modelos de custo para técnicas como estruturas de acesso, particionamento, e reescrita de consultas.

6.2.1 Cálculo do custo e benefício de soluções combinadas

A versão do algoritmo da mochila proposto para o passo de seleção testa todas os arranjos possíveis entre as soluções combinadas geradas. A cada iteração, o algoritmo testa o benefício total e o peso total da mochila **com** e **sem** a solução sendo avaliada. Durante esta avaliação, é necessário considerar além das ações que compõem a solução sendo avaliada, as ações das soluções que já estão dentro da mochila. Diferentemente do cenário tradicional, uma ação pode estar em mais de uma solução. Logo, o custo de criação deve ser contabilizado uma vez, mas o benefício é somado para cada solução combinada.

Considere o exemplo da Figura 6.1. As soluções S1 e S2 possuem o mesmo conjunto de ações A1 e A2, enquanto S3 compartilha apenas A1 com S1 e S2. O peso das soluções S1 e S2 é a soma dos recursos necessários para implementar

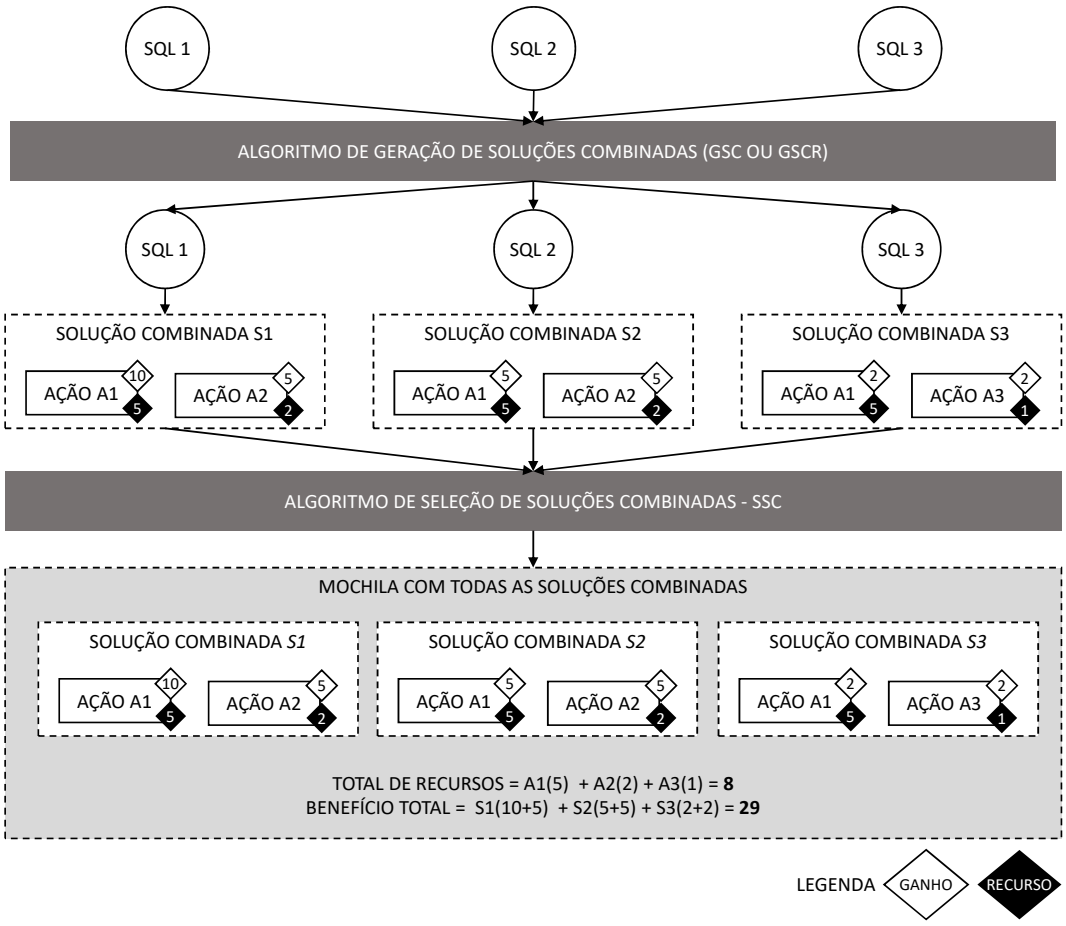


Figura 6.1: Exemplo da etapa de seleção de soluções combinadas para ilustrar o modelo de custo proposto. Consideram-se duas ações para cada solução apenas para simplificá-lo. Não existe um número fixo de ações para cada solução e podem possuir diferentes quantidades.

A1 e A2 (5+2) uma única vez, já que são as mesmas ações e serão executadas uma única vez. Já o benefício é a soma do benefício gerado pelas ações A1 e A2 nos comandos SQL 1 e SQL 2: (10+5)+(5+5). Em outras palavras, criam-se as ações uma vez apenas, mas uma ação pode beneficiar mais de um comando e integrar múltiplas soluções combinadas.

No exemplo da Figura 6.1, as três soluções geradas são selecionadas como conjunto final. O total de recursos necessários para implementar as ações (peso da mochila) é a soma de recursos de todas as ações (A1, A2, A3) que integram as soluções (5+2+1=**8**). Já o benefício total (valor da mochila) é a soma dos benefícios de cada ação em relação à cada comando SQL da carga de trabalho: (10+5+5+5+2+2=**29**).

6.3 Resumo do capítulo

Neste capítulo, apresentou-se uma proposta de algoritmo para seleção automática de soluções combinadas que escolhe o conjunto de soluções combinadas que minimizam o custo de execução da carga de trabalho e respeitam os recursos computacionais disponíveis. Este algoritmo foi modelado a partir do problema da mochila baseado em programação dinâmica. Esta Abordagem é independente das técnicas envolvidas na geração das soluções, e independente também do SGBD em questão. Também apresentou-se como calcular o custo e benefício de soluções combinadas. O próximo capítulo apresenta a implementação do método de sintonia fina global e dos algoritmos de geração e seleção de soluções combinadas.

7
Implementação

Este capítulo apresenta a implementação do método de sintonia fina global proposto. Descreve e justifica as principais escolhas de tecnologias.

Uma possível implementação para os especialistas de sintonia fina modelados são agentes de software [90, 91, 6]. Um sistema multiagentes pode gerar respostas coletivas e complexas através das iterações entre componentes individuais relativamente simples. Essa abordagem permite projetar o comportamento macroscópico usando apenas as interações dos agentes locais, e permite que o sistema mude seu comportamento sem interferência externa explícita durante o tempo de execução [92].

Baseado nisso, propomos a implementação de especialistas de sintonia fina como agentes de software. São unidades de software assíncronas, independentes, autocontidas, e que representam diferentes algoritmos para geração e seleção de soluções combinadas. Eles são agentes interativos, autônomos, direcionados por metas e adaptáveis. Cada um tem um papel único no sistema e podem ser instanciados de acordo com as preferências expressas pelo administrador do banco de dados.

A escolha de agentes de software pode ser justificada pela extensa literatura que explora, entre outros, características desejáveis para o método de sintonia fina global como auto-organização e assincronicidade.

Auto-organização: um determinado sistema multiagente pode ser classificado como auto-organizado quando consegue organizar seus agentes sem direção, manipulação ou controle externos [93]. O Comitê de especialistas não usa nenhum controle central, interno ou externo explícito para organizar os especialistas. Isso permite a modelagem de um sistema de auto-organização forte e capaz de combinar soluções usando um ou N especialistas. Utilizando agentes auto-organizados, o número de especialistas que compõem o comitê pode ser definido em tempo de instanciação, com base nos indivíduos selecionados pelo DBA no início da tarefa de sintonia fina.

Assincronicidade: Os especialistas executam diferentes algoritmos internos para gerar soluções combinadas, com diferentes heurísticas e tempos de execução. Isso faz com que o tempo de resposta seja diferente entre eles a cada passo do algoritmo de geração. No entanto, eles precisam executar as iterações

dos algoritmos GSC e GSCR que tomam como entrada as soluções combinadas geradas na iteração anterior.

Mesmo que a busca em largura permita uma execução paralela dentro de uma iteração, existe uma serialização no modo que as soluções são combinadas pelo algoritmo *branch-and-bound*. Para resolver este problema, propomos o uso de um ciclo de execução assíncrono para cada especialista através da utilização do *blackboard* (Seção 5.3.2) [84]. Isso viabilizou a modelagem de um comportamento em que, a cada determinado intervalo de tempo executa a seguinte sequência de regras:

i Consulta o *blackboard* por estados de soluções combinadas que aquele especialista ainda não contribuiu;

ii Propõe soluções combinadas combinadas ou independentes para aquele estado;

iii Grava um novo estado no *blackboard* para que outros especialistas possam contribuir em futuras iterações do GSC ou GSCR.

Note que o comportamento de combinação diz respeito ao passo de avaliação de um nó dos algoritmos GSC e GSCR. É um comportamento que é executado independentemente de quantos ou quais especialistas compõem o comitê, e isso dá a liberdade de uma execução tanto assíncrona quanto distribuída. Uma vez tratados os desafios de coordenar uma ação através de uma rede não confiável (como falhas bizantinas [94]), os especialistas podem ser executados em diferentes nós da rede. dividindo o custo de processamento e aumentando a capacidade de análise. Consequentemente, o espaço de busca analisado na busca pela solução ótima não foi explorada neste trabalho, mas é uma opção a ser estudada futuramente.

7.1 O sistema multiagentes

Dois ou mais especialistas (agentes de software nesta implementação) acordados em realizar uma tarefa de sintonia fina formam o comitê de especialistas. O foco desta tese é no método de geração e seleção de soluções combinadas, mas existem outros passos necessários para a execução de uma tarefa de sintonia fina, como a captura da carga de trabalho e execução das soluções selecionadas. Foi necessária a implementação de dois agentes para realizar estas tarefas extras: o agente Observador e Executor (Figura 7.1).

O Observador é responsável pela aquisição e tratamento de informações usadas como entrada para os algoritmos GSC e GSCR (1 na Figura 7.1).

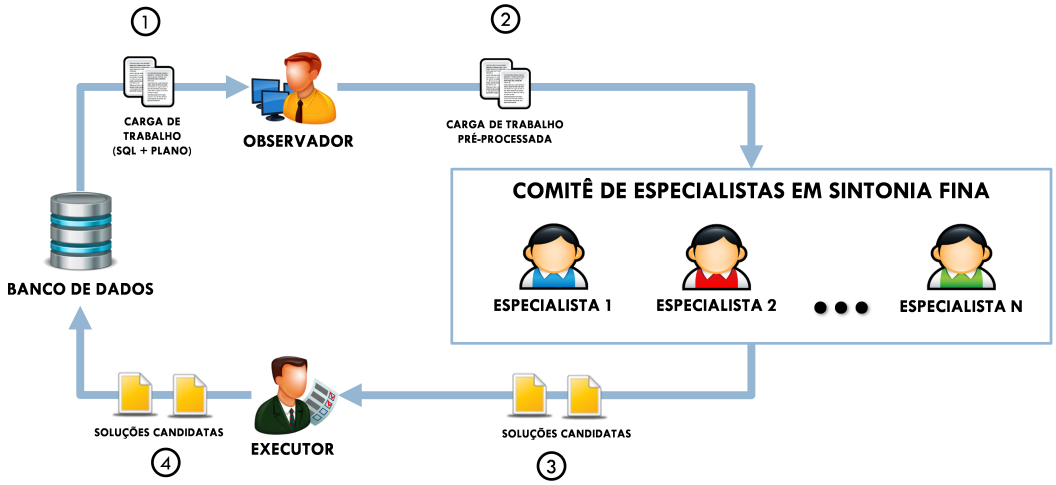


Figura 7.1: Sistema multiagente de software para execução de uma tarefa de sintonia fina global

Entre as informações adquiridas estão os comandos SQL da carga de trabalho, seus respectivos planos de execução, e dados da metabase do banco com estatísticas sobre tabelas, atributos e dados armazenados. Sobre a tratamento, faz-se necessário a extração de dados do comando SQL (até então no formato texto) para que, independente do SGBDR utilizado, possa-se trabalhar sob um mesmo formato de dados e nomenclatura.

O Comitê de Especialistas é formado pelo conjunto selecionado no início da execução pelo DBA. Dentre os especialistas implementados, o DBA é livre para escolher qualquer combinação de elementos para a formação do comitê. Também é necessário escolher qual versão do algoritmo de geração será usada para realizar as combinações – o algoritmo GSC ou sua versão relaxada GSCR.

O Executor, como o nome sugere, executa as soluções filtradas pelo algoritmo SSC (3 na Figura 7.1). O Executor também é responsável por escolher o melhor momento para aplicar as soluções combinadas, convencionalmente em momentos em que o banco de dados tem baixa demanda como períodos noturnos, ou em uma janela de tempo pré-determinada pelo DBA.

A Figura 7.1 ilustra o fluxo de execução do sistema multiagente proposto. O processo se inicia com o agente Observador capturando a carga de trabalho e metadados do banco (1 na Figura 7.1). A carga é analisada e pré-processada em três passos pelo observador: i) extração de dados do comando SQL (até então no formato texto); ii) extração de dados do plano de execução; e iii) extração de estatísticas do banco de dados, e em seguida estes dados são enviados ao comitê de especialistas (1 na Figura 7.1). O comitê executa um dos algoritmos de geração propostos GSC ou GSCR e, ao fim da fase de geração, o algoritmo de seleção SACC é aplicado no conjunto de todas as soluções combinadas gerando um subconjunto de soluções que minimizam o custo de execução da carga de

trabalho e respeitam as restrições de recursos computacionais (3 na Figura 7.1). Por fim, o Executor aplica o conjunto final de soluções combinadas no banco de dados (4 na Figura 7.1).

7.2 Os especialistas que compõem o comitê

Para a avaliação do método de sintonia fina global, implementou-se três especialistas a partir de três algoritmos de técnicas relevantes e conhecidas. As técnicas usadas foram visões materializadas, índices completos, e índices parciais. Esta escolha se deu pela natureza comum entre as três técnicas, que são estruturas de acesso utilizadas para reduzir o custo de leitura durante a execução de comandos SQL. Isso permitiu uma combinação natural entre elas do que seria entre outras técnicas como gerência de buffer, particionamentos e outras.

Entende-se que a combinação não se restringe a apenas estruturas de acesso e que, teoricamente, nada impede combinar estruturas de acesso com outras técnicas. Inclusive faz parte dos trabalhos futuros a implementação de especialistas para outros tipos de ações como particionamento horizontal e gerência de parâmetros. Porém, limitou-se o escopo nesta tese a três especialistas que puderam ser modelados, implementados e testados dentro da restrição de tempo da pesquisa. São eles:

ESP_VMA_19: Agente especialista em geração de visões materializadas. O algoritmo de Agrawal et al. [20] foi estendido para a geração de soluções combinadas dentro do processo de geração dos algoritmos GSC e GSCR. A geração de VMAs é realizado através da reescrita do comando SQL dado como entrada, e de estatísticas da metabase do banco de dados. Outra significativa extensão foi a inclusão de suporte a consultas com subconsultas. Estendeu-se a estratégia do algoritmo para, na presença de consultas com subconsultas, gerar visões materializadas para a(s) subconsulta(s) e deixar que o comando externo seja sintonizado por outras técnicas como índices parciais e completos. A hipótese é que, minimizando o custo da subconsulta, o comando todo será beneficiado, e a não reescrita do comando externo trará possibilidades de combinação com outras estratégias, gerando mais possibilidades de combinação. Exemplos apresentados no capítulo de experimentos mostram a efetividade desta proposta, e que a hipótese de que soluções combinadas podem gerar melhores benefícios para a carga de trabalho pode ser verdadeira.

ESP_IND_19: Agente especializado em geração de índices completos. O algoritmo de Monteiro [6] foi estendido em dois sentidos: a) para gerar soluções combinadas usando os algoritmos GSC e GSCR, e b) propor índices sobre

visões materializadas. Na versão original do algoritmo, os índices são propostos apenas sobre tabelas que existem fisicamente no banco de dados. Neste trabalho o algoritmo precisa também propor índices sobre visões materializadas que ainda não foram criadas no banco, o que obrigou a adaptação dos modelos de custos responsáveis pelo cálculo do custo de criação da estrutura, e do ganho hipotético da estrutura.

ESP_PIN_19: Agente especializado na geração de índices parciais. Este especialista foi criado a partir do algoritmo de Dominguez [52] que propõe um algoritmo de geração e um modelo estatístico para escolher os melhores critérios de restrição para índices parciais. Assim como o algoritmo de Monteiro, o algoritmo de Dominguez também foi estendido em dois aspectos: a) para gerar soluções combinadas usando os algoritmos GSC e GSCR, e b) propor índices sobre visões materializadas. No caso dos índices parciais, além da adaptação do algoritmo para escolha do atributo de uma visão materializada, tivemos que estender também a escolha de atributos de restrição desta mesma VMA.

Todos os agentes foram implementados utilizando JAVA SE, e o JAVA Agent DEvelopment Framework (JADE) [95]. JADE é uma plataforma de software livre para aplicações de agentes ponto-a-ponto na linguagem Java. Este *framework* simplifica a implementação de sistemas multiagentes através de um *middleware* que atende às especificações internacionais da Foundation for Inteligent Physical Agents (FIPA), uma organização IEEE Computer Society [96]. Trata-se de uma série de padrões para aplicações baseadas em agentes que permite a interoperabilidade e interação com outras tecnologias. A utilização destes padrões acelerou o desenvolvimento e garantiu uma significativa qualidade de código para o protótipo.

7.3 Extensão para três SGBDRs

Para avaliação do método de sintonia fina global proposto, e para gerar evidências de sua generalidade, a ferramenta implementada foi estendida para três SGBDs do mercado:

- SQL Server 2017 Developer [97];
- Oracle 12c [98];
- PostgreSQL 9.7 [99];

A extensão para diferentes SGBDs foi realizada sem grandes desafios de programação, principalmente pela modelagem de especialistas como unidades

autônomas, e o encapsulamento de diferentes modelos de custo que compartilham as mesmas interfaces. Todos os pontos de código estendidos foram planejados previamente durante a modelagem dos métodos de geração e seleção de soluções combinadas.

O principal desafio foi a adaptação dos modelos de custo para as especificidades de cada SGBD. De forma geral, resumiu-se a encontrar os comandos SQL correspondentes para a consulta de informações da metabase do banco de dados. O PostgreSQL possui um documento extraoficial [100] que descreve como extrair e interpretar informações da metabase utilizadas na tarefa de sintonia fina. Foi publicado em 2007 e referencia a versão 8.2 do PostgreSQL, mas ainda pode ser usado como guia após atualizações pontuais. Já para o SQL Server e Oracle, foi necessário um trabalho de investigação utilizando a documentação oficial e fóruns de discussão na internet, onde DBAs compartilham experiências e detalhes de implementação.

7.4 Resumo do capítulo

Neste capítulo descrevemos brevemente a implementação dos especialistas de sintonia fina utilizando agentes de software. Também descrevemos a implementação e fluxo de execução do sistema multiagentes com ênfase no comitê de especialistas. São listados os três especialistas implementados para a avaliação do método e como os algoritmos originais foram estendidos para o contexto de soluções combinadas. No próximo capítulo, apresentaremos os resultados experimentais utilizando a implementação aqui descrita.

8
Experimentação

Este capítulo apresenta os experimentos realizados para avaliar o método de sintonia fina global proposto. Propomos diferentes cenários de teste em diferentes SGBDs na tentativa de gerar evidências sobre a eficácia e eficiência do método de sintonia fina global. A Seção 8.1 apresenta as configurações de hardware usadas. As métricas usadas estão descritas na Seção 8.2, e o *benchmark* usado na Seção 8.3. Os cenários de teste para cada experimento estão descritos na Seção 8.4. Os resultados e análises estão descritos na Seção 8.5, onde são feitas avaliações segundo a) os cenários, b) os modelos de consulta do *benchmark*, c) exemplos específicos de soluções combinadas, d) entre os algoritmos GSC e GSCR, e) e uma comparação com outras ferramentas de sintonia fina.

8.1
Setup

O hardware usado para testes foi uma máquina servidora, com processador Intel Core I7-7700 de 3.60GHz, 32GB RAM, Disco Rígido (HDD) de 1TB. O sistema operacional usado foi Windows Server 2016 64 bits.

Para evitar a concorrência por recursos entre o SGBD e a ferramenta de sintonia fina, com introdução de ruídos nos resultados, utilizou-se uma máquina cliente independente para rodar a ferramenta. O hardware da máquina cliente não influenciou nos resultados, uma vez que o processamento das consultas e estatísticas relevantes apresentadas aqui vieram do servidor. Entretanto, para fins de reprodutibilidade, possui a seguinte configuração: processador Intel I7-7820 2.9GHz, 12GB de memória RAM, HDD de 512GB, e sistema operacional Windows 10 64bits.

8.2
Métricas

A métrica utilizada para comparar a performance nos diferentes cenários foi a unidade de custo de execução de consultas do otimizador do SGBD utilizado durante o teste. Esta unidade de custo é um valor arbitrário de cada SGBD, mas normalmente é uma função entre dois valores: i) o custo de ler as

Tabela	Número de Tuplas	Tamanho em disco	% Tamanho total
lineitem	59986100	15 GB	67.6%
orders	15000000	4.175 MB	18.3%
partsupp	8000000	2.008 MB	8.8%
part	2000000	776 MB	3.4%
customer	1500000	385 MB	1.7%
supplier	100000	33 MB	0.1%
nation	25	24 kB	0.0%
region	5	24 kB	0.0%
TOTAIS	86586130	22GB	100%

Tabela 8.1: Estatísticas sobre o banco de dados do TPC-H usado nos experimentos.

páginas de disco necessárias para executar a consulta, e ii) o custo de CPU para processar os dados em memória principal.

8.3 Benchmark

Os benchmarks de banco de dados são amplamente utilizados para testar o desempenho das ferramentas de sintonia fina. Para testar a implementação do método proposto, utilizando o TPC Benchmark™ H (TPC-H) [101]. Foi criado pelo Transaction Processing Performance Council (TPC), uma organização sem fins lucrativos cujo objetivo principal é estabelecer critérios para obtenção de informações sobre o desempenho de bancos de dados através do uso de *benchmarks* [102].

O TPC-H consiste em um conjunto de consultas *ad-hoc* orientadas a negócios. As consultas e os dados que preenchem o banco de dados foram escolhidos para ilustrar sistemas de suporte à decisão que examinam grandes volumes de dados, executam consultas com alto grau de complexidade e dão respostas a questões críticas de negócios.

O tamanho do banco de dados do TPC-H pode ser configurado durante a instalação. Os tamanhos disponíveis variam entre 1GB e 100TB e se referem ao tamanho dos dados brutos, que podem ocupar um espaço maior após inseridos no banco e criação de chaves primárias e estrangeiras. Possui um conjunto de 22 modelos de consultas que podem ser instanciadas com diferentes valores nos atributos de restrição. Estes valores são escolhidos aleatoriamente dentro da faixa possível de cada atributo. Logo, apesar de aleatórios, se restringem às tuplas que existem no banco e, normalmente, toda consulta instanciada a partir destes modelos retorna pelo menos uma tupla como resposta.

Para os experimentos descritos aqui, gerou-se um banco de dados com 10GB de tamanho. Após a inserção dos dados no banco e criação das chaves

primárias e estrangeiras, este volume cresceu para 22GB. A Tabela 8.1 apresenta detalhes sobre as tabelas que compõe o banco, como número de tuplas, tamanho, e proporção em relação ao banco. Ela é importante para a explicação de exemplos de soluções combinadas discutidas a seguir.

Em relação às consultas, foram instanciadas 30 unidades distintas de cada um dos 22 modelos, totalizando 660 consultas. Dado a complexidade das consultas, considerou-se que 10GB seria uma massa de dados suficientes para evidenciar a eficácia e eficiência do método, e permitiria a execução de um número maior de testes.

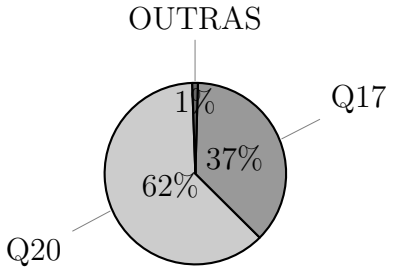


Figura 8.1: Distribuição do custo de execução das consultas sem nenhuma ação de sintonia fina - Cenário ORIGINAL

Duas consultas instanciadas a partir do mesmo modelo podem ter diferentes custos de execução, já que possuem diferentes valores de restrição nos atributos da cláusula *where*. Além disso, os diferentes modelos de consultas (Anexo A) possuem uma natural variação no custo de execução. No entanto, existe um grande desbalanceamento entre as consultas do TPC-H. Os modelos Q7 e Q20 geram as consultas mais custosas da carga de trabalho.

O Gráfico 8.1 mostra as distribuições dos custos de execução das 660 consultas geradas agrupadas por modelos. As 60 consultas dos modelos Q20 e Q17 somadas correspondem a 99% do custo de executar toda a carga de trabalho, enquanto as outras 600 consultas, geradas a partir dos outros 20 modelos, correspondem a apenas 1% do custo total. Este desbalanceamento é importante para explicar os resultados a seguir.

Para o Benchmark TPC-H em cenários sem nenhuma ação de sintonia fina, o custo de obter páginas em disco geralmente domina o custo de CPU, e o custo de executar uma consulta é em grande parte explicado por operações de leitura em memória secundária. Apesar de ser uma carga analítica que frequentemente aplica operadores matemáticos aos dados, as consultas possuem muitas junções e varreduras completas de tabelas.

8.4 Cenários

Para execução dos experimentos, testou-se diferentes combinações de especialistas para a formação do comitê. Os cenários testados foram:

(a) ***ORIGINAL***: Custos originais de todas as consultas sem nenhuma ação de sintonia fina. Apenas os índices primários, chaves primárias e estrangeiras sugeridas pelo *benchmark* foram criadas no banco de dados.

(b) ***IND***: apenas o especialista em índices completos é ativado.

(c) ***PIN***: apenas o especialista em índices parciais é ativado.

(d) ***VMA***: apenas o especialista em visões materializadas é ativado.

(e) ***IND+PIN***: especialistas em índice completo e índice parcial são ativados.

(f) ***IND+PIN+VMA***: especialistas em índices completos, índices parciais e visões materializadas são ativados.

(g) ***(VMA+IND)***: Especialistas em visões materializadas e índices completos são ativados, mas apenas as soluções combinadas que possuem visões materializadas indexadas por índices completos são consideradas, não são criados índices em outras estruturas além de VMAs.

(h) ***(VMA+PIN)***: Especialistas em visões materializadas e índices parciais são ativados, mas apenas as soluções combinadas que possuem visões materializadas indexadas por índices parciais são consideradas, não são criados índices em outras estruturas além de VMAs.

(i) ***(VMA+IND+PIN)***: Especialistas em visões materializadas, índices completos, e índices parciais são ativados, mas apenas as soluções combinadas que possuem visões materializadas indexadas por índices completos e/ou parciais são consideradas, não são criados índices em outras estruturas além de VMAs.

(j) ***IND + PIN + (VMA+IND+PIN)***: Todos os especialistas são considerados, e todas as soluções combinadas válidas são consideradas durante os processos de geração e seleção.

8.4.1 Metodologia de testes

Para cada um dos cenários, (exceto o ORIGINAL que executa apenas o passo v), os seguintes passos foram executados para a avaliação:

(i) O(s) especialista(s) selecionado(s) para a tarefa de sintonia fina é(são) ativado(s);

(ii) As 660 consultas instanciadas são executadas em ordem aleatória;

(iii) O(s) especialista(s) observa(m) a carga de trabalho e gera(m) e seleciona(m) as soluções combinadas;

(iv) As soluções combinadas selecionadas são executadas no banco de dados;

(v) As 660 consultas instanciadas são executadas em ordem aleatória e os custos de cada execução são coletados.

8.5 Resultados e Análise

Esta seção apresenta os resultados dos experimentos utilizando a implementação do método proposto. Os resultados estão agrupados por cada um dos SGBDs utilizados nos testes.

Esta seção apresenta os resultados usando o PostgreSQL e todos os cenários propostos. Primeiro apresenta-se uma análise geral que compara os diferentes cenários, depois mostra-se o desempenho detalhado de cada modelo de consulta de consulta em cada cenário. Por fim, apresenta-se uma comparação entre os algoritmos GSC e GSCR em relação ao número de soluções combinadas geradas e o desempenho de cada um deles.

8.5.1 Análise dos resultados no escopo dos cenários

Esta seção discute a comparação entre todos os cenários de testes propostos. Os resultados são a soma dos custos de execução de todas as consultas da carga de trabalho para cada cenário em relação ao cenário ORIGINAL. Figura 8.2 mostra que todas as combinações de especialistas trouxeram benefícios para a carga de trabalho. É possível ainda verificar que o especialista em índices é o mais eficiente.

Nos cenários onde apenas um especialista está ativo - IND, PIN, VMA - o melhor desempenho foi alcançado pelo especialista em índices completos IND, com custo de execução total equivalente a 1% do cenário ORIGINAL,

ou seja, um benefício de 99%.. O segundo melhor foi o especialista em índices parciais PIN com 37% do custo ORIGINAL, e por último VMA com 73%. Já os cenários onde as soluções combinadas foram possíveis pela presença de dois ou mais especialistas, em todos os casos a combinação trouxe benefícios para a carga de trabalho.

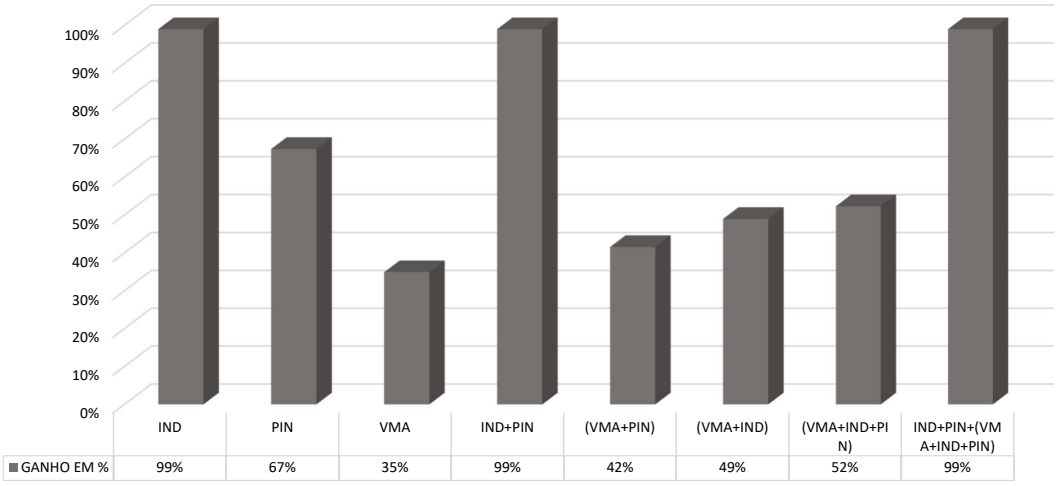


Figura 8.2: Custo de execução para cada cenário em relação ao custo original.

A Figura 8.3 mostra os três melhores cenários da Figura 8.2 onde mostra o custo de execução absoluto para ilustrarmos a diferença entre eles, uma vez a proximidade nos benefícios obtidos.

É visível que com o aumento das possibilidades de combinação, os resultados melhoram. Mesmo que o especialista mais eficiente seja o de índice completo, a inclusão dos outros dois na geração de soluções combinadas trouxe benefícios para a carga de trabalho. Como evidência podemos verificar que dentre todas as combinações testadas, o melhor desempenho foi do cenário IND+PIN+(VMA+IND+PIN) - aquele que aceita todos os tipos de combinações, pelo menos para o TPC-H.

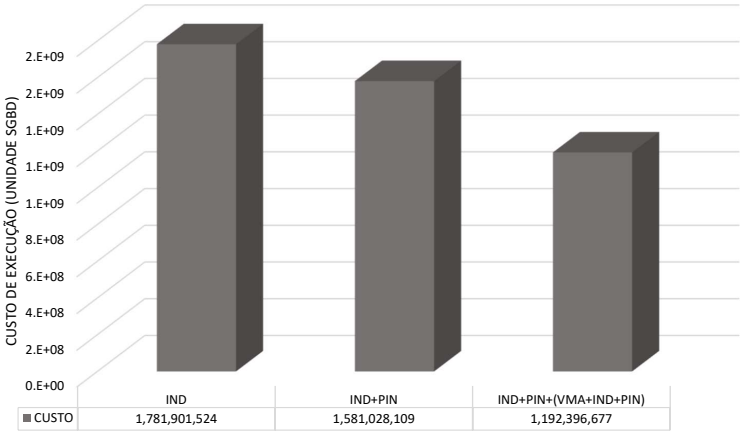
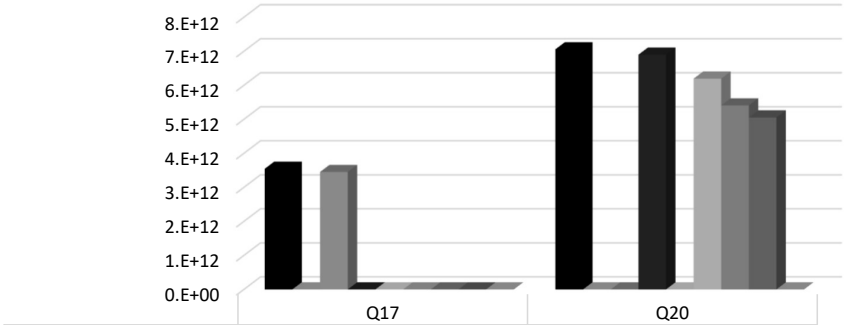


Figura 8.3: Comparação do custo de execução entre os melhores cenários testados.

8.5.2
Análise dos resultados no escopo dos modelos de consulta

Para a avaliação de cada cenário em relação aos 22 modelos, dividimos os resultados em dois gráficos de acordo com a ordem de grandeza dos custos. Os custos para os modelos mais custosos Q17 e Q20 são apresentados na Figura 8.4. Para as demais consultas, a Figura 8.5.

A primeira observação em relação ao desempenho do método é o ótimo desempenho na redução de custos das consultas Q17 e Q18 (Figura 8.4) quando todas as ações de combinação foram consideradas - cenário IND+PIN+(VMA+IND+PIN). Nota-se que a maior redução de custos foi proveniente às soluções geradas pelo especialista de índices completos, o cenário em que apenas o especialista IND_monteiro_00 está ativado (IND) mostra redução de aproximadamente 99% do custo de execução tanto para a Q17 quanto Q20, assim como todos os outros em que ele participa do comitê de especialistas - IND+PID, IND+PIN+(VMA+IND+PIN).



	Q17	Q20
ORIGINAL	3,544,818,720,566	7,056,212,214,671
IND	236,940,364	71,288,708
PIN	3,451,936,573,247	835,142,289
VMA	90,451,860	6,899,260,472,039
IND+PIN	259,839,983	36,922,444
(VMA+PIN)	90,448,364	6,195,345,811,410
(VMA+IND)	90,447,029	5,408,338,783,400
(VMA+IND+PIN)	90,447,029	5,054,108,478,953
IND+PIN+(VMA+IND+PIN)	34,834,448	41,922,444

Figura 8.4: Custo de execução para cada cenário agrupado pelas famílias de consultas Q17 e Q20.

Ainda considerando os resultados da Figura 8.4, outro desempenho relevante foi do especialista VMA_agrawal_00. A criação de uma visão materializada para o modelo Q17 também foi capaz de reduzir o custo de execução em 99%. Em contraste, a solução combinada que possui apenas visão materializada (VMA) para o modelo Q20 não teve o mesmo desempenho, e gerou um custo próximo ao original. O melhor cenário da Q20 foi o

IND+PIN+(VMA+IND+PIN) com todas as combinações entre todos os especialistas.

A Figura 8.5 mostra o desempenho de cada cenário agrupados pelos modelos de consulta. Nota-se, novamente, que os cenários onde o especialista em IND está ativado tiveram bons desempenhos. Alguns modelos como Q12, Q16 também foram significativamente melhorados pelo especialista em VMA. De modo geral, o cenário com maior redução no custo de execução das consultas foi o que considera todas as combinações possíveis IND+PIN+(VMA+IND+PIN).

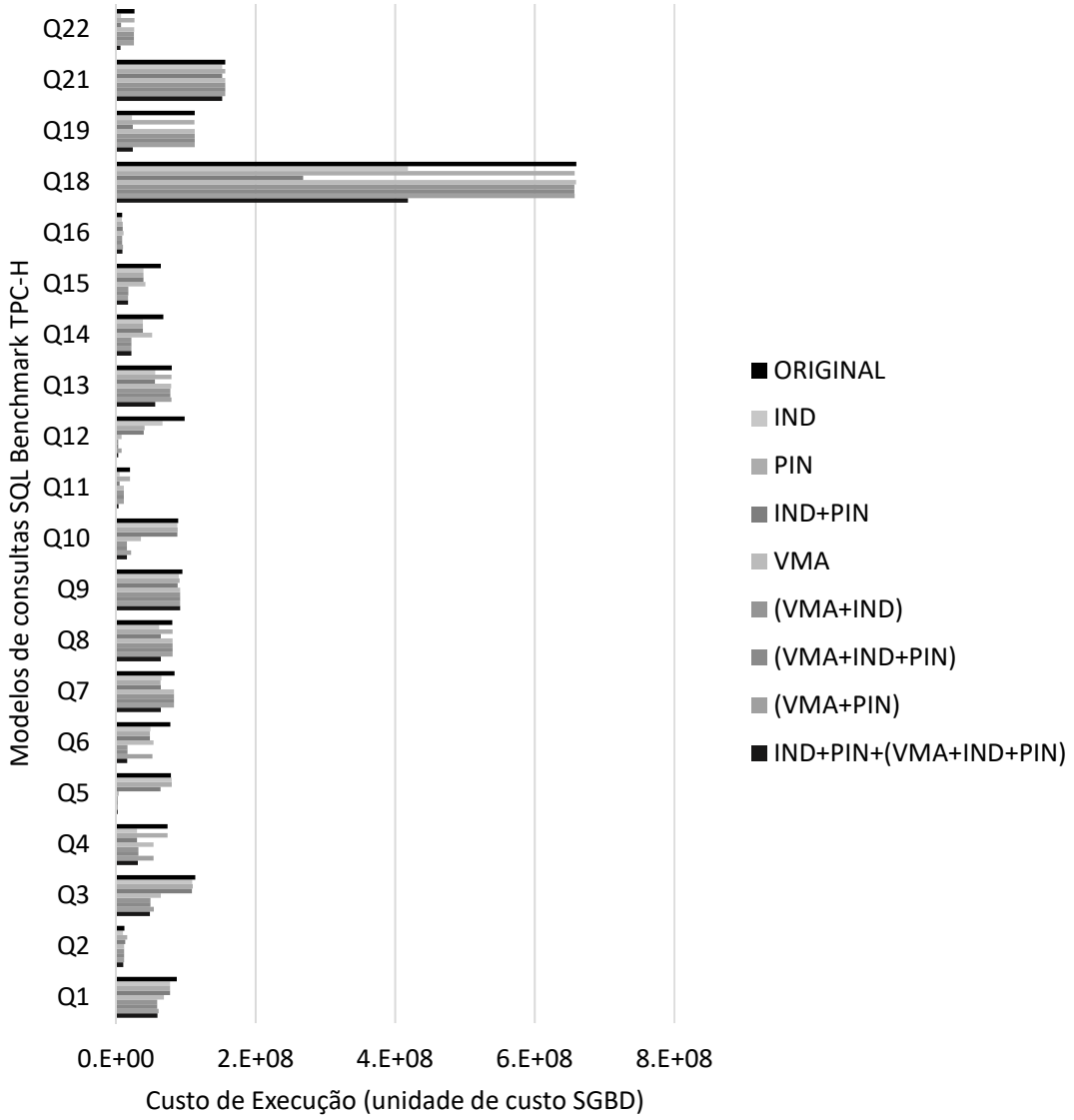


Figura 8.5: Custo de execução para cada cenário agrupado por família de consultas.

Para simplificar a comparação entre os cenários ORIGINAL e IND+PIN+(VMA+IND+PIN), apresenta-se os Gráficos das Figuras 8.6 e 8.7. Os gráficos mostram também o valor equivalente de executar a consulta

em relação ao valor original. Por exemplo, na Figura 8.6 executar as consultas do modelo Q1 no cenário IND+PIN+(VMA+IND+PIN) corresponde a 68% do custo do cenário ORIGINAL, logo, uma redução de 32% utilizando as soluções combinadas.

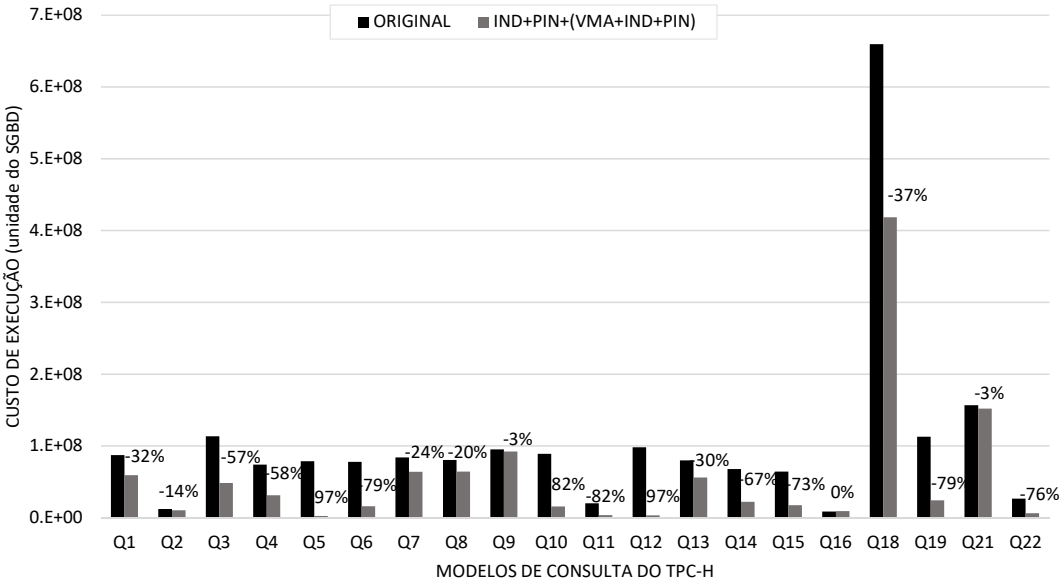


Figura 8.6: Custo de execução para cenários ORIGINAL e IND+PIN+(VMA+IND+PIN) agrupado pelas consultas com menor custo de execução.

De forma geral, o Figura 8.6 mostra que as soluções combinadas trouxeram benefício para a maioria das consultas da carga de trabalho. A única exceção foi a consulta Q16 que, pelo pequeno custo de execução, a ferramenta optou por não sugerir nenhuma solução combinada.

Por fim, a Figura 8.7 mostra o principal caso de sucesso das soluções combinadas. Neste exemplo, incluímos os valores absolutos para ilustrar o ganho em relação à execução do cenário ORIGINAL. Em ambas as consultas, soluções combinadas foram as responsáveis pela redução obtida.

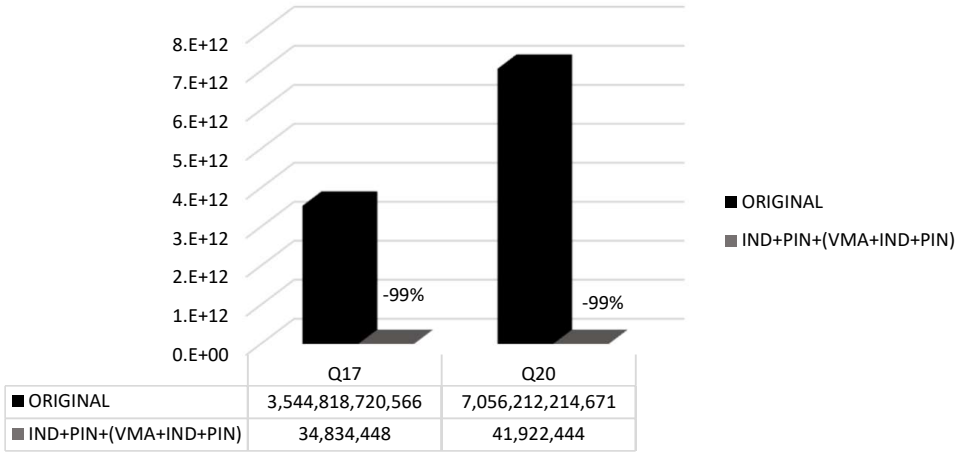


Figura 8.7: Custo de execução para cenários ORIGINAL e IND+PIN+(VMA+IND+PIN) agrupado pelas consultas com maior custo de execução.

8.5.3 Análise a partir de exemplos de soluções combinadas

O desempenho das soluções combinadas para reduzir o custo de execução das instâncias de consultas dos modelos Q17 e Q20 é uma evidência da eficácia das estratégias de combinação e seleção. Além disso, outros modelos de consultas também geraram soluções combinadas interessantes. Nas seções a seguir, discutimos três exemplos de soluções combinadas para a Q4, Q11 e Q17.

8.5.3.1 Exemplo Q4

Discutimos aqui exemplos para ilustrar as soluções combinadas geradas e selecionadas pelo método proposto. O primeiro exemplo selecionado foi a instância 10 da consulta Q4 apresentado abaixo:

```sql
/* TPC_H Q4_instancia_10 */
SELECT o_orderpriority,
       count( * ) AS order_count
FROM orders
WHERE o_orderdate >= date '1998-07-25'
  AND o_orderdate < date '1998-07-25' + 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;
```

Segundo a documentação do TPC-H, a consulta Q4 foi modelada para determinar a eficiência do sistema fictício de pedidos simulado pelo *benchmark*, filtrando pedidos que não foram entregues em atraso (*l_commitdate* < *l_receiptdate*). O plano de execução desta consulta sem nenhuma ação de sintonia fina pode ser visto na Figura 8.8. Note que as operações de maior custo são uma varredura completa na tabela *lineitem*, a maior tabela do banco que corresponde a 67% do tamanho total do banco de dados (Tabela 8.1), seguida de outra varredura completa na tabela *orders* - segunda maior tabela com 18%.

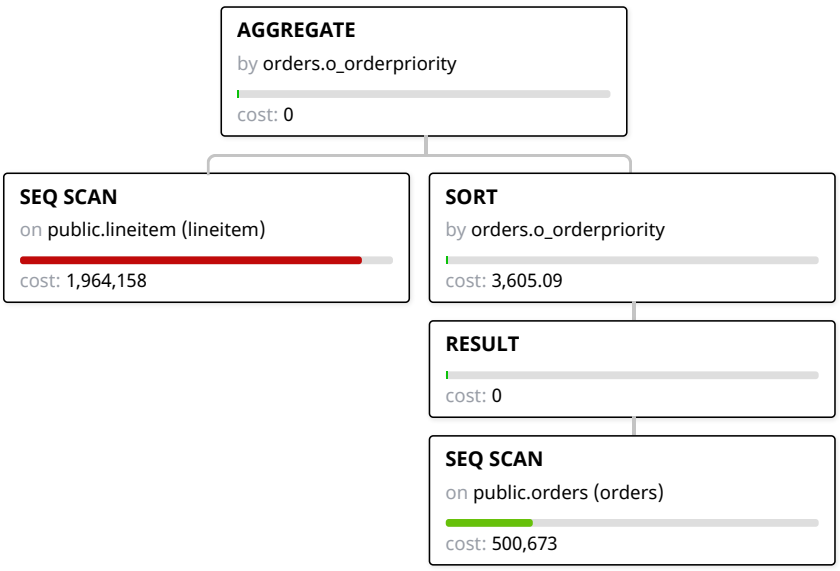


Figura 8.8: Plano de execução instância da *Q4_instancia_10* - Cenário ORIGINAL - sem nenhuma ação de sintonia fina.

Para este exemplo *Q4_instancia_10* de consulta e seu plano, o método proposto gerou e selecionou a seguinte solução candidata:

```
1  /* SOLUCAO COMBINADA para TPC_H Q4_instancia_10 */
2  /* ACAO 1 */
3  CREATE materialized VIEW MV_TAP_Q4 AS
4  SELECT * FROM lineitem WHERE l_commitdate < l_receiptdate;
5
6  /* ACAO 2 */
7  CREATE INDEX ID_TAP_S_N914717080 ON mv_tap_q4(l_orderkey);
8
9  /* ACAO 3 */
10 CREATE INDEX PID_TAP_N285721920 ON orders (o_orderdate)
11 WHERE orders.o_orderdate >= '1998-07-25'
12   AND orders.o_orderdate < '1998-10-25';
```

Esta solução combinada possui três ações. A ACÃO 1 consiste em uma visão materializada para a subconsulta da Q4 que filtra as tuplas da tabela *lineitem* que foram entregues dentro do prazo. Note que a visão materializada *MV_TAP_Q4* pode ser usada para qualquer instância Q4.

A AÇÃO 2 sugere a criação de um índice para a visão materializada proposta na ACÃO 1. Neste exemplo, o algoritmo identificou que a coluna *l_orderkey* é usada como filtro pela consulta externa (linha 10 da consulta), e portanto, uma oportunidade para obter tuplas da VMA sem a necessidade de uma varredura completa através de um índice completo.

Por fim, a ACÃO 3 sugere a criação de um índice parcial específico para a *Q4_instancia_10* onde um índice parcial indexa a coluna *o_orderdate* usada pela consulta externa para filtrar a tabela *orders*.

O plano de execução da *Q4_instancia_10* após a execução da solução combinada anterior pode ser visto na Figura 8.9. O otimizador de consultas substituiu as duas varreduras completas do plano original por varreduras usando os índices propostos pela solução combinada.

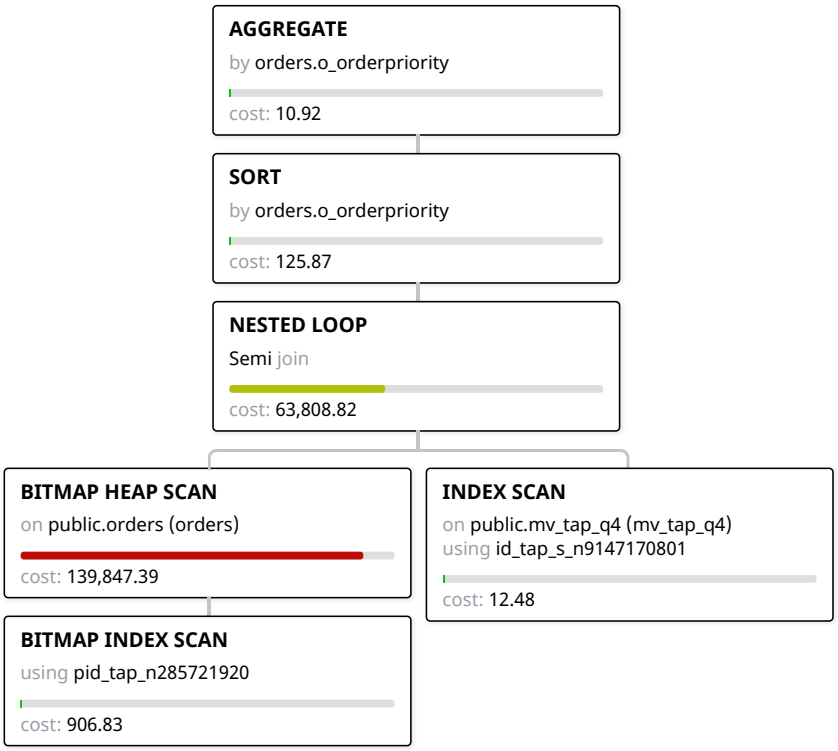


Figura 8.9: Plano de execução instância da *Q4_instancia_10* - Cenário IND+PIN+(VMA+IND+PIN).

Vale salientar que a abordagem aqui apresentada foi capaz de propor ações tanto para a consulta externa, quanto para a subconsulta deste exemplo. Normalmente as abordagens de sintonia fina da literatura tem dificuldades para lidar com subconsultas. A estratégia de percorrer o espaço de busca por força bruta permite que os diferentes especialistas testem um número grande de permutações entre suas ações de forma sistemática. Se por um lado o custo de testar este número de combinações pode ser dispendioso e precisa ser controlado, por outro permite soluções complexas que talvez um DBA humano teria dificuldades em planejar e testar.

8.5.3.2 Exemplo Q11

Outro exemplo de solução combinada foi gerada para a instância 5 do modelo Q11 apresentada abaixo. Apesar de não ser um dos comandos mais caros da carga de trabalho.

```sql
/* TPC_H Query Q11_instancia_5 */
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 = 'VIETNAM'
GROUP BY PS_PARTKEY
HAVING SUM(PS_SUPPLYCOST*PS_AVAILQTY) >
  (SELECT SUM(PS_SUPPLYCOST*PS_AVAILQTY) * 0.0001000000 AS
     cost_sum
   FROM PARTSUPP, SUPPLIER, NATION
   WHERE PS_SUPPKEY = S_SUPPKEY
     AND S_NATIONKEY = N_NATIONKEY
     AND N_NAME = 'INDIA')
ORDER BY VALUE DESC;
```

O modelo Q11 também possui uma subconsulta, e foi beneficiada pela mesma heurística de reescrita de consultas proposta aqui e executada pelo especialista *VMA_OLIVEIRA_19*. Para esta instância, foi gerada a seguinte solução combinada:

```
/*  SOLUCAO COMBINADA para TPC_H Q11_instancia_5 */
/*ACAO 1*/
CREATE materialized VIEW MV_TAP_Q11 AS
SELECT sum(ps_supplycost*ps_availqty) * 0.0001000000 AS
   cost_sum,
       partsupp.ps_suppkey,
       supplier.s_suppkey,
       supplier.s_nationkey,
       nation.n_nationkey,
       nation.n_name
FROM partsupp,
     supplier,
     nation
WHERE ps_suppkey = s_suppkey
  AND s_nationkey = n_nationkey
GROUP BY partsupp.ps_suppkey,
         supplier.s_suppkey,
         supplier.s_nationkey,
         nation.n_nationkey,
         nation.n_name;

/*ACAO 2*/
CREATE INDEX ID_TAP_S_N1201801232 ON
                    MV_TAP_Q11 (n_name, n_nationkey);
/*ACAO 3*/
CREATE INDEX ID_TAP_S_N720851507 ON
                    supplier (s_nationkey);
/*ACAO 4*/
CREATE INDEX ID_TAP_S_1060061719 ON
                    partsupp (ps_suppkey);
```


Note que a ACAO 1 *MV_TAP_Q11* foi escrita para ser genérica o suficiente para responder todas as instância do modelo Q11. Qualquer país usado na restrição *N_NAME* da cláusula *where* ('INDIA' na *Q11_instancia_5*), a *MV_TAP_Q11* consegue responder. As únicas restrições na cláusula *where* da VMA são de chaves estrangeiras, apenas para garantir a correção do resultado.

Além da VMA, a solução combinada proposta para a *Q11_instancia_1* possui três outras ações, mais especificamente, três índices completos. A AÇÃO 2 é um índice composto nas colunas *n_name, n_nationkey* da VMA *MV_TAP_Q11* proposta pela AÇÃO 1. As ações 3 e 4 são índices nas tabelas *supplier* coluna *s_nationkey* e *partsupp* coluna *ps_suppkey* respectivamente. Mais uma vez, uma solução combinada que sugere uma VMA+IND para uma subconsulta, e índices para otimizar a consulta externa do comando SQL.

A Figura 8.10 apresenta o plano de execução sem nenhuma ação de sintonia fina, e a Figura 8.11 mostra o plano de execução criado pelo otimizador do SGBD após a execução da solução combinada.

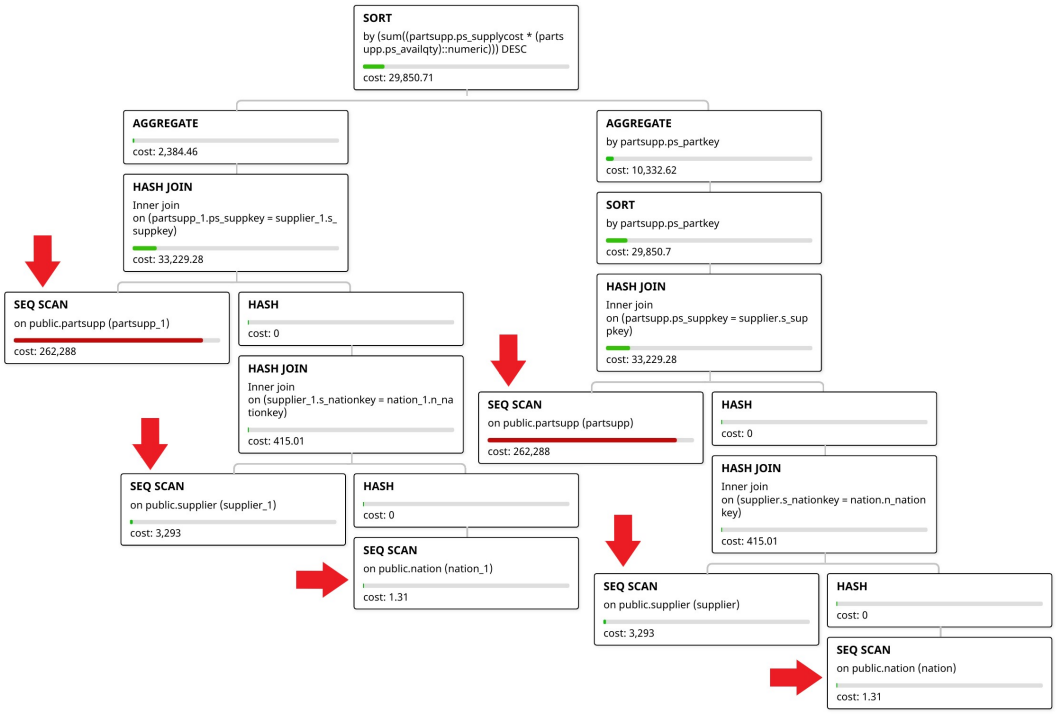


Figura 8.10: Plano de execução da consulta Q11_instancia_1 no cenário ORIGINAL. O otimizador de consultas planejou 6 varreduras completas (marcadas com setas).

O primeiro ponto a ser observado no plano do cenário ORIGINAL é a quantidade de varreduras completas: 2 na *partsup*, 2 na *supplier*, 2 na *nation*. Dentre as 6 varreduras, 4 foram nas tabelas *nation* e *supplier*, e não trouxeram grande custo. Porém as duas varreduras na tabela *partsupp* foram significativas, já que ela é a terceira maior do banco (Tabela 8.1).

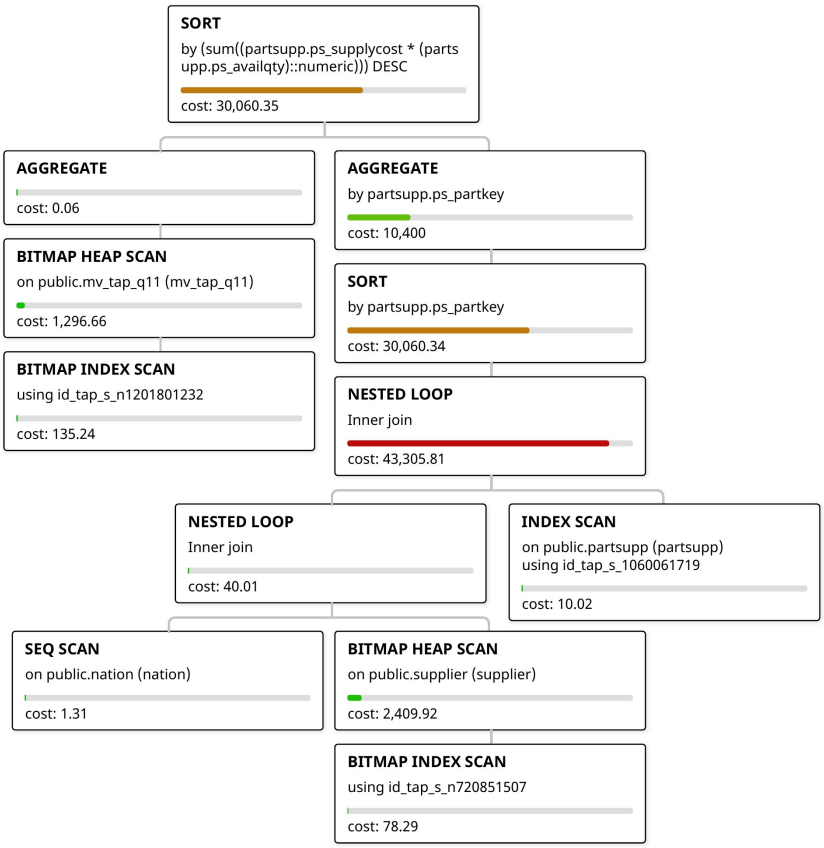


Figura 8.11: Plano de execução da consulta Q11_instancia_1 no Cenário IND+PIN+(VMA+IND+PIN) usando uma visão materializada indexada, e outros dois índices para obtenção dos dados. Apenas a tabela *nation* continua com uma varredura completa, mas ela possui 25 tuplas, logo não foram sugeridas soluções para ela.

A Figura 8.11 mostra o plano de execução no cenário IND+PIN+(VMA+IND+PIN) e um plano completamente diferente do cenário ORIGINAL. Neste plano uma das varreduras da tabela *partsupp*, que correspondia à execução da subconsulta foi substituída pela visão materializada *MV_TAP_Q11* que por sua vez foi lida usando o índice *ID_TAP_S_N1201801232*. Já na parte externa da consulta, a segunda varreduda completa da *partsupp* foi substituída por um *index scan* utilizando o *ID_TAP_S_N720851507*. Por fim, a varredura da tabela *supplier* foi executada por um *BITMAP HEAP SCAN* usando o *ID_TAP_S_N720851507*. O única varredura completa foi executada na tabela *nation* porque, como foi discutido no Capítulo 5, o algoritmo não gera soluções combinadas para tabelas muito pequenas, e a *nation* possui apenas 25 tuplas o que caracterizaria uma solução inexpressiva.

Para concluir este exemplo: o método proposto gerou uma solução combinada que evitou a varredura completa das tabelas mais importantes do

comando, evitando propor soluções para estruturas com tamanhos desprezíveis em relação às outras. Comando este que possui uma subconsulta que dificulta a geração de soluções automáticas. Por fim, a operação mais custosa do plano se tornou o *nested loop* em decorrência do *inner join*, que custou 43.305. Note que no custo original, apenas uma das (duas) varreduras completas da tabela *partsupp* custava 262.288, mais de 6 vezes maior.

Novamente os detalhes dos números evidenciam a eficiência do método, mas o principal ponto do experimento está no fato que uma solução combinada como VMA, que por si só já traria benefício, teve seu desempenho melhorado. Em outras palavras, a ação de sintonia fina foi combinada com outras ações que se complementam no escopo do mesmo comando SQL. Entende-se que essa é uma evidência importante para a eficácia da estratégia automática de geração e seleção de soluções globais proposta aqui.

8.5.3.3 Exemplo Q17

O último exemplo é a instância 2 da consulta Q17 apresentada abaixo. Vale ressaltar que as consultas instanciadas a partir do modelo Q17 correspondem a 37% do custo total de executar a carga de trabalho, e tratam-se de consultas com alto custo de execução.

```
1  /* TPC_H Query Q17_instancia_2 */
2  SELECT
3      SUM(L_EXTENDEDPRICE)/7.0 AS AVG_YEARLY
4  FROM LINEITEM,
5       PART
6  WHERE P_PARTKEY = L_PARTKEY
7    AND P_BRAND = 'Brand#12'
8    AND P_CONTAINER = 'JUMBO BAG'
9    AND L_QUANTITY <
10     (
11     SELECT
12         0.2*AVG(L_QUANTITY) AS avg_quantity
13     FROM LINEITEM
14     WHERE L_PARTKEY = P_PARTKEY
15     )
```

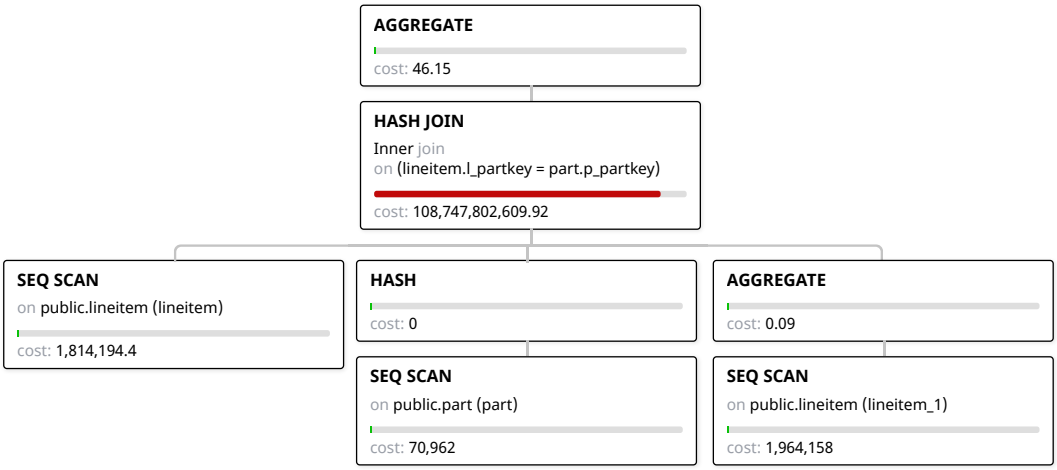


Figura 8.12: Plano de execução da *Q17_instancia_2* - Cenário ORIGINAL

A Figura 8.12 mostra o plano de execução da *Q17_instancia_2* sem nenhuma ação de sintonia fina. Já a Figura 8.13 mostra o cenário IND+PIN+(VMA+IND+PIN) onde os três especialistas são livres para tentar qualquer combinação entre as ações. No plano do cenário ORIGINAL, o SGBD realizou uma varredura completa na tabela *part* e duas varreduras completas na tabela *lineitem*. Já no plano do cenário IND+PIN+(VMA+IND+PIN) a varredura da *lineitem* correspondente à subconsulta foi substituída pela visão materializada *ma_tap_q17* e a parte externa da consulta foi otimizada por dois índices, um completo na tabela *lineitem* e outro parcial na tabela *part* (Figura 8.13).

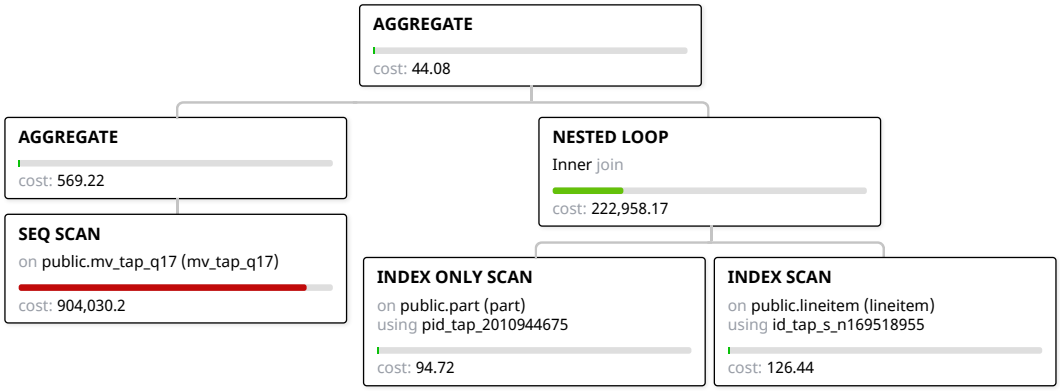


Figura 8.13: Plano de execução instância da *Q17_instancia_2* - Cenário IND+PIN+(VMA+IND+PIN)

Esta solução mostrou-se efetiva em diferentes casos porque devido à estratégia de combinação, espera-se que tratando o desempenho da subconsulta através de VMAs, o desempenho da consulta externa seja tratado por outras técnicas que não precisam reescrever o comando SQL, como índices. É um

exemplo da estratégia de *dividir para conquistar*, em que conseguimos beneficiar comandos complexos pela iteração entre os especialistas envolvidos.

Só para comparar a ordem de grandeza dos custos entre os planos de execução da **Q17_instancia_1**, podemos citar o maior custo no cenário IND+PIN+(VMA+IND+PIN) que é a varredura completa da *mv_tap_q17* de aproximadamente 904 mil (Figura 8.13), isso é metade do valor de uma das duas varreduras da tabela *lineitem* no plano ORIGINAL (Figura 8.12). A diferença entre eles é tão grande que, o maior custo do cenário ORIGINAL é de aproximadamente 108 bilhões de unidades de custo do PostgreSQL.

Note que neste exemplo da consulta *Q17_instancia_1*, o custo de execução foi reduzido significativamente por uma solução combinada entre três técnicas diferentes que se complementam.

Independente dos números envolvidos e da eficiência da solução combinada, e mesmo que o ganho não tivesse sido tão grande, o fato mais importante é que este exemplo ilustra a **eficácia** do método antes da eficiência. O complemento entre as diferentes técnicas aqui não é aleatório, foi sistematicamente modelado a partir de um uma estratégia de sintonia fina global e gerado por uma ferramenta automática.

8.5.4 Comparação entre os algoritmos GSC e GSCR

Todos os resultados apresentados nas seções anteriores foram gerados a partir do algoritmo GSCR. Apresenta-se nesta seção uma comparação entre os dois algoritmos de geração GSC e GSCR propostos nesta tese.

A principal diferença entre o GSC e GSCR é a regra de poda implementada. Enquanto o GSC expande apenas o melhor estado de cada iteração, o GSCR expande todos os estados que melhoram o custo original de execução do comando SQL. Consequentemente, o GSCR percorre um espaço maior de busca e pode eventualmente pode produzir melhores soluções. A Figura 8.14 mostra uma comparação entre o GSC e GSCR usando o cenário IND+PIN+(VMA+IND+PIN).

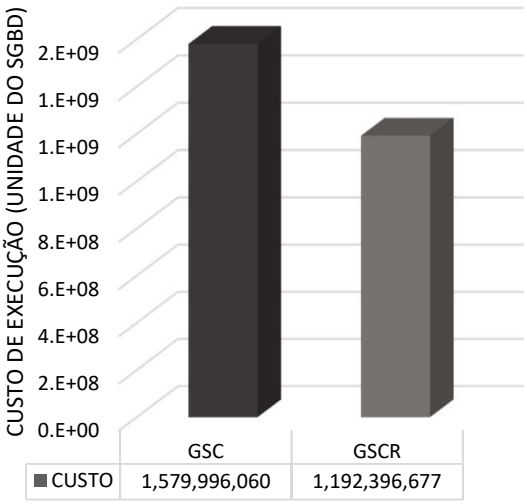


Figura 8.14: Custo de execução total entre os algoritmos GSC e GSCR usando o cenário IND+PIN+(VMA+IND+PIN). Quanto menor o custo de execução, melhor.

O GSCR conseguiu um desempenho melhor que o algoritmo GSC no cenário testado. O algoritmo GSCR surgiu como uma alternativa à imprecisão que os modelos de custo externos ao SGBD podem possuir. Esperava-se que explorando um maior espaço de busca o método pudesse encontrar melhores combinações, o que aconteceu na prática como podemos verificar na Figura 8.14. Porém, investigando a diferença entre o desempenho dos dois algoritmos, notou-se que o algoritmo GSC, mesmo sendo executado no cenário IND+PIN+(VMA+IND+PIN) que permite a combinação com visões materializadas, produziu um número muito menor de soluções combinadas que possuíam VMAs que a versão GSCR.

Para demonstrar a diferença que a falta de visões materializadas produziram no desempenho do GSC, compara-se o resultados dos dois algoritmos com o cenário IND+PIN, onde apenas as combinações entre índices completos e índices parciais são possíveis. Os resultados são mostrados na Figura 8.15, e é possível nota que o algoritmo GSC teve um desempenho muito semelhante ao cenário que não usa VMAs durante a combinação.

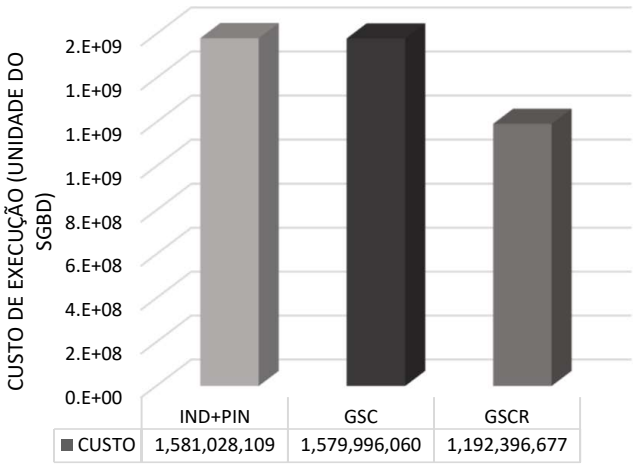


Figura 8.15: Custo de execução total entre os algoritmos GSC e GSCR e o cenário IND+PIN

Após uma avaliação do passo de geração usando o GSC, descobriu-se que o motivo de poucas soluções combinadas possuírem VMAs não era apenas por uma deficiência do modelo de custo como se imaginava inicialmente, mas também pelas regras de poda dos algoritmos propostos.

No exemplo da Figura 8.16, o algoritmo GSC só expandirá o estado 1 com a VMA quando tal solução possuir maior benefício que os estados 2 e 3 (regra de poda do GSC). Isso significa que, caso o estado 2 ou 3 que sugerem IND e PIN respectivamente tenham um ganho maior que a VMA, a sub-árvore do estado 1 será podada e combinações a partir do estado 1 não serão avaliadas.

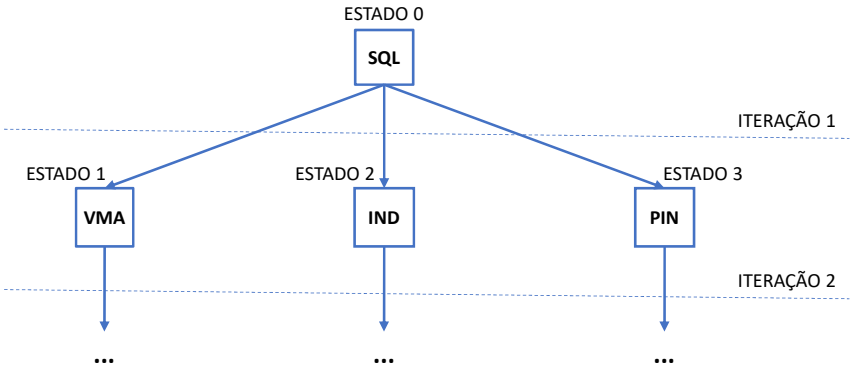


Figura 8.16: Exemplo de execução do algoritmo GSC e GSCR.

Para a carga de trabalho avaliada, as visões materializadas, maioria dos casos, são melhores alternativas que índices apenas quando são indexadas. Quando o algoritmo GSC compara as soluções da iteração 1, os índices ainda não foram propostos para a VMA, logo, as soluções com visões materializadas indexadas não são avaliadas porque são podadas antes da combinação com índices. E isso explica os desempenhos similares entre o algoritmo GSC e o cenário IND+PIN.

8.5.5 Comparação com outras ferramentas de sintonia fina

Como parte da avaliação, os resultados do método são comparados com outras duas ferramentas de sintonia fina para o SGBD PostgreSQL, chamadas Dexter [60] e PoWA [62]. A Figura 8.17 mostra o ganho de cada cenário testado em relação a estas duas ferramentas.

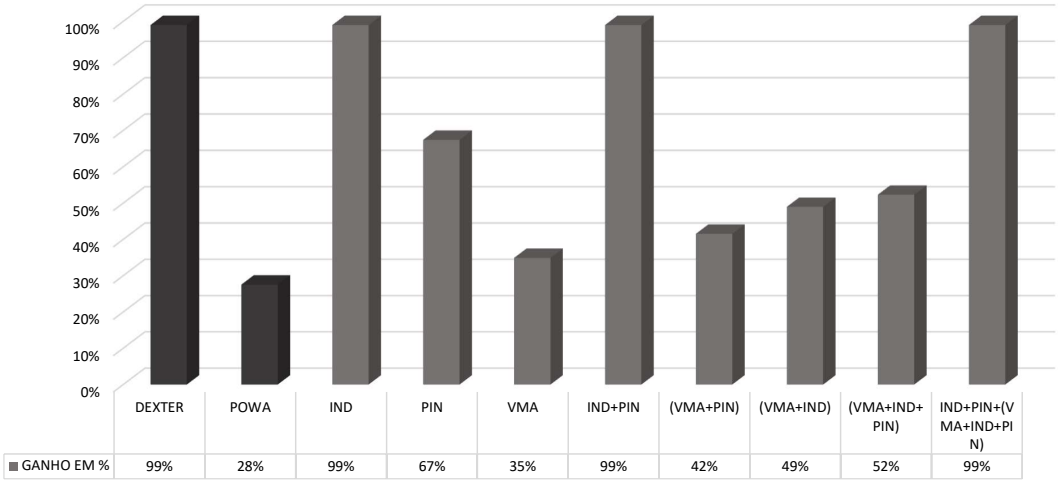


Figura 8.17: Ganho total de cada cenário em relação às ferramentas Dexter e PoWA. Quanto maior o ganho, melhores foram as ações de sintonia fina.

Os resultados da ferramenta Dexter são muito competitivos. Assim como os cenários IND, IND+PIN, e IND+PIN+(VMA+IND+PIN) a Dexter conseguiu uma redução de 99% do custo de execução da carga de trabalho. Já a ferramenta PoWA teve o pior resultado de todos os testes. Para entender melhor a diferença de desempenho entre os melhores cenários, a Figura 8.18 apresenta os custos originais dos melhores.

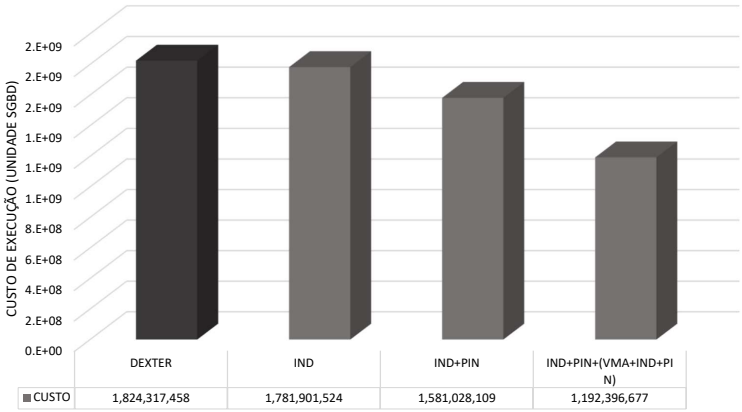


Figura 8.18: Custo de execução total dos melhores cenários e das ferramentas Dexter e PoWA. Quanto menor o custo, melhor o desempenho.

Os custos mostram que o método proposto aqui foi melhor em todos os cenários em relação à ferramenta Dexter. Nota-se que mesmo o cenário IND onde, assim como o Dexter também usa índices completos, a estratégia de geração e seleção proposta teve um desempenho melhor para a carga de trabalho usada. Para os demais cenários onde foram consideradas outras técnicas como IND+PIN a diferença cresceu, e no cenário onde todas as combinações possíveis entre os especialistas foram permitidas, o custo de executar a carga de trabalho foi 34% menor em relação ao Dexter.

8.6 Resumo do Capítulo

Este capítulo apresentou os resultados experimentais do método de sintonia fina global proposto aqui. Descrevemos os detalhes dos experimentos, como máquina usada, *benchmark* TPC-H, massa de dados, e consultas geradas para a carga de trabalho. Discutimos os resultados a partir de diferentes perspectivas: uma análise dos resultados no escopo i) dos cenários propostos, ii) dos modelos de consulta do TPC-H, iii) de alguns exemplos de soluções combinadas. Por fim, apresentou-se uma comparação entre os algoritmos GSC e sua versão relaxada GSCR, e uma comparação com outras ferramentas de sintonia fina. De forma geral, os resultados mostraram que as soluções combinadas trouxeram benefícios para a carga de trabalho, que o método de sintonia fina global é eficaz e eficiente nos cenários testados. O próximo Capítulo discute o método proposto.

9
Discussão sobre o método

Apresenta-se aqui, uma discussão sobre a extensão do método proposto. A Seção 9.1 discute sobre como estender o método de combinação proposto, e a Seção 9.2 apresenta os principais desafios de selecionar ações de sintonia fina para cargas de trabalhos com atualizações.

9.1
Extensão do método de combinação

Uma vez definido o método de combinação das técnicas de sintonia fina, discutimos aqui os possíveis passos necessários para extensão do método para outros tipos de ações.

Para incluir um novo método no processo de combinação, podemos listar três passos possíveis em alto nível: i) identificar formas de integrar as técnicas de sintonia, ii) eliminar soluções mutuamente exclusivas, e iii) eliminar soluções inválidas e inexpressivas.

9.1.1
Identificar formas de integrar as técnicas de sintonia

A combinação de técnicas precisa ser planejada e explicitamente modelada nos algoritmos de geração. Métodos automáticos de combinação podem gerar ações não previstas inicialmente, ou emergentes, mas a integração entre as diferentes técnicas, assim como o que pode ou não pode ser combinado, é um conhecimento que precisa ser modelado no algoritmo de geração.

Técnicas que compartilham a mesma estratégia como estruturas de acesso são boas alternativas para a combinação. Quando duas técnicas têm o mesmo objetivo, reduzir o custo de varredura de tabelas por exemplo, pode-se buscar formas de combinar ações de modo que a solução final produza ganhos equivalentes, mas com um custo de implementação menor.

Por exemplo, índices completos e parciais podem se complementar indexando diferentes colunas da mesma relação ou relações diferentes [52]. Esta similaridade pode ser explorada em uma abordagem de combinação. Um índice completo traz o benefício de se evitar uma varredura completa na tabela ao executar alguma operação e, consequentemente, reduzir tanto a quantidade

de leituras lógicas como o custo de processamento. Já o índice parcial possui os mesmos benefícios mas apenas em circunstâncias como a) existe uma diferenciação dos dados ativos e não ativos, ou b) existe termos específicos de busca que particionam o conjuntos de dados. No entanto, o índice parcial tem um custo de criação e atualização menor que o índice completo, uma vez que indexa apenas um subconjunto das tuplas de uma tabela, através da sua expressão condicional. Logo, traz ganhos equivalentes (ou maiores) que um índice completo, mas com um custo menor de implementação nos casos específicos que pode ser aplicado.

Neste exemplo, uma forma de integrar índices completos e índices parciais seria tratar os casos gerais com a técnica mais custosa e genérica (índices completos) e casos específicos com soluções menos dispendiosas (índices parciais). Note que esta conclusão foi possível apenas após a identificação das formas de integração pelo estudo das técnicas, e não uma enumeração completa das soluções possíveis. Quando se tenta combinar técnicas como seleção de índices onde, para geração de um único tipo de índice, já é provado ser um problema NP-Difícil [44], identificar as formas de integração e modelá-las no algoritmo de geração é imperativo para métodos automáticos.

9.1.2 Eliminar soluções mutuamente exclusivas

Dadas duas soluções mutuamente exclusivas i e v, seria um desperdício de recursos computacionais a execução de ambas as soluções. Seria um desperdício do custo de criação, do custo de manutenção (se houver), e dos recursos computacionais consumidos – por exemplo, memória secundária para persistência. Uma vez que os recursos computacionais são finitos, identificar e evitar tais competições durante o processo de geração traz dois benefícios: primeiro, permite distribuir melhor as soluções combinadas para cobrir uma maior quantidade de comandos da carga de trabalho e, segundo, reduz o espaço de busca das combinações possíveis entre as técnicas de sintonia fina.

Diferentemente das estratégias de integração que são modeladas no algoritmo de geração, métodos para identificar e eliminar soluções mutuamente exclusivas são modeladas no algoritmo de seleção. Na fase de geração, os comandos SQL são analisados caso a caso, e soluções combinadas são geradas no escopo de um único comando SQL. Já a concorrência entre soluções não implica inviabilidade das ações, mas um desperdício de recursos computacionais que pode ser identificado globalmente pela fase de seleção. Em outras palavras, a integração das técnicas cria soluções mais eficazes e pertence ao escopo de geração, enquanto a identificação de conflitos analisa e elimina alternativas

redundantes (eficientes) e pertence ao escopo de seleção.

9.1.3 Eliminar soluções inválidas e inexpressivas

Eliminar soluções inválidas é importante para evitar erros durante a execução das soluções selecionadas. Eliminar soluções inexpressivas é importante para a redução do espaço de busca durante a fase de seleção. Ambas são classificações ligadas a limites tecnológicos e algoritmos internos do SGBD.

As premissas de a) integração das técnicas e b) eliminação das soluções mutuamente exclusivas descrevem a lógica de como soluções podem ser combinadas e filtradas para qualquer SGBD. Já a premissa de c) eliminação das soluções inválidas e soluções inexpressivas está no âmbito da implementação dos SGBDs e precisa ser modelada individualmente para cada um deles, pois são regras que impedem que soluções inexequíveis naquele contexto específico sejam geradas ou selecionadas pelo método automático de sintonia fina global.

9.2 Cenários de uso e atualizações na carga de trabalho

Durante a avaliação deste trabalho, considerou-se uma carga de trabalho OLAP sem atualizações (inserções, atualizações, e deleções) ou atualizações pontuais e com uma frequência muito baixa.

O método de combinação apresentado aqui possui um modelo de custo que não considera atualizações, mas entendemos que mesmo assim existem cenários significativos que o método pode ser aplicado. Por exemplo, sistemas de banco de dados que armazenam operações financeiras de anos fiscais fechados, para armazenamento de logs, aplicações baseados em eventos históricos, ou ainda aplicações como *blockchain* onde dados são adicionados ao repositório de modo linear e cronológico, e não sofre alterações após a inclusão.

Autores dedicam na literatura trabalhos inteiros de pesquisa para explorar o custo de atualizar estruturas de acesso e como este custo impacta na escolha de quais estruturas criar no banco de dados [103, 104, 39, 105, 29, 106]. Nicolas Bruno tem dois capítulos em seu livro *Automated Physical Database Design and Tuning* [3] que tratam muito bem do problema de manter estruturas de acesso automaticamente e continuamente em um banco de dados.

Em particular para visões materializadas, a tese de doutorado de Gupta [83] descreve em detalhes o desafio de selecionar VMAs na presença de atualizações da carga de trabalho. Ele também apresenta como alternativa um modelo de custo baseado em benefícios e penalidades para, dada uma carga de trabalho, predizer o ganho de uma VMA e o custo para mantê-la. Também

propõe um conjunto de políticas de atualizações para VMAs que minimizam este custo de atualização.

Para estruturas do tipo índices, a tese de doutorado de Monteiro [6] também descreve uma heurística chamada heurística de benefícios (proposta anteriormente [5, 14]) como uma alternativa para identificar: (a) índices que o custo de atualização supera o benefício e devem ser excluídos do banco, e (b) índices que o benefício supera o custo de atualização mas devido as atualização estão armazenados de forma fragmentada no disco causando um custo de leitura maior que o necessário e, portanto, precisam de uma operação de manutenção para serem desfragmentados.

O método aqui proposto não foi avaliado para cargas com atualizações, mas isso não significa que ele não possa considerar tais casos. Note que entre os elementos de entrada dos algoritmos de geração propostos GSC e GSCR, e do algoritmo de seleção SSC, está o modelo de custo para avaliação das soluções combinadas. Entende-se que a atualização na carga de trabalho impacta a avaliação sobre a utilidade, ou eficácia, da estrutura de acesso. Para o método proposto aqui, essa eficácia não é definida pelos algoritmos propostos, mas pelo modelo de custo, que essencialmente foi modelado como um componente externo ao método.

Nesta tese, não consideramos um modelo de custo que avalie, além do ganho da solução combinada, a penalidade gerada pela atualização das estruturas propostas. Mas entendemos que quando tivermos este modelo desejável, teoricamente, isso não impactará no método proposto visto que apenas este componente dado como entrada para os algoritmos será diferente do atual. Logo, faz parte dos trabalhos futuros contemplar também cargas de trabalho com atualizações.

O trabalho de Morelli [14] apresenta uma estratégia para lidar com o custo de manutenção de estruturas de acesso para cargas de trabalho com atualizações, onde para cada estrutura calcula-se o ônus e o bônus de criá-la e mantê-la no banco de dados. Estruturas de acesso em que o ônus é maior que o bônus são eliminadas, ou caso sejam índices e estejam fragmentados no disco, executa-se uma operação de desfragmentação do índice. Este trabalho de Morelli é um candidato possível para extensão e implementação de um modelo de custo que também considere a penalidade gerada por atualizações e, quando estendido, poderá ser usado pelo método de combinação aqui proposto.

9.3 Resumo do capítulo

Apresenta-se aqui um estratégia em alto nível de como estender o método de combinação proposto. Também discutimos sobre cargas de trabalho com atualizações e o motivo de não termos avaliado este cenário. O próximo Capítulo apresenta as conclusões da pesquisa desta tese.

10
Conclusão

O trabalho de sintonia fina de banco de dados é uma tarefa importante, e é caracterizada pela complexidade de se encontrar um bom conjunto de ações que minimizam o custo de execução da carga de trabalho e/ou aumentam a vazão (*throughput*) de transações. Neste contexto, o problema se torna ainda mais complexo quando tenta-se combinar diferentes técnicas, que por sua vez possuem diferentes algoritmos e heurísticas.

Neste cenário, realizar a tarefa de sintonia fina de forma manual pode ser árduo e complicado. Não existe na literatura, nenhum método abrangente e automático para a geração de soluções combinadas durante uma única tarefa de sintonia fina.

Investigou-se nesta tese um método para gerar combinações de ações válidas a partir de diferentes técnicas de sintonia fina, e uma vez geradas, como selecionar tais soluções combinadas na presença de restrições tecnológicas e limites de recursos computacionais de forma a minimizar o custo de execução da carga de trabalho. Nas seções a seguir, apresentamos as contribuições desta tese.

10.1
Contribuições principais

Entende-se que a principal contribuição científica desta tese, que avançam o estado da arte é:

(i) um método de combinação de ações de sintonia fina independente de técnicas capaz de:

- gerar ações combinadas entre diferentes técnicas percorrendo o espaço de busca aplicando restrições devido à complexidade de se testar todas as alternativas.
- selecionar através de uma estratégia global o conjunto de soluções combinadas que minimizam o custo de execução da carga de trabalho e respeitam os limites dos recursos computacionais disponíveis.

10.2 Contribuições secundárias

Em paralelo às contribuições principais, também alcançou-se contribuições tecnológicas e extensões à estratégias de sintonia fina propostas na literatura:

(a) *Modelagem de um sistema multiagentes.* Para o desenvolvimento dos especialistas, foi utilizada uma abordagem baseada em agentes de software autônomos, auto-organizados, e assíncronos. Apesar de não ser uma contribuição para a área de engenharia de software, é uma aplicação inédita da teoria de agentes para o problema de combinação de ações de sintonia fina em bancos de dados.

(b) *Extensão de três algoritmos de sintonia fina.* Os algoritmos de Agrawal [20], Monteiro [6] e Dominguez [52] que originalmente geravam ações locais, foram estendidos para a geração também de soluções combinadas. Em particular, os algoritmos de índices foram adaptados para indexar visões materializadas, além de tabelas do banco de dados.

(c) *Reescrita de subconsultas para geração de visões materializadas.* O algoritmo de Agrawal [20] foi estendido para reescrever subconsultas e gerar visões materializadas. Essa estratégia beneficia soluções combinadas por tratar as subconsultas e as consultas externas do mesmo comando SQL usando técnicas diferentes.

(d) *Extensão de modelos de custos para soluções combinadas.* Assim como os algoritmos de geração, foi necessário o desenvolvimento de um modelo de custo específico para avaliar soluções combinadas. O modelo proposto aqui foi criado a partir de três modelos da literatura [26, 6, 52], que separadamente predizem custos para visões materializadas, índices completos e índices parciais. Este trabalho unificou estes modelos e estendeu-os para contemplar visões materializadas indexadas por índices completos e índices parciais.

(e) *Implementação do método de combinação.* Para avaliação do método proposto, implementou-se uma ferramenta de sintonia fina automática extensível em três dimensões: i) inclusão de novas técnicas, ii) inclusão de novos algoritmos para técnicas já suportadas, iii) inclusão de suporte a novos SGBDs. Na versão corrente da ferramenta algoritmos para IND, PIN, e VMA estão implementados, assim como suporte aos SGBDs PostgreSQL, Oracle, e SQL Server.

10.3 Oportunidades para Trabalhos Futuros

Devido à limitação de tempo de uma tese de doutorado, não foi possível explorar todas as possibilidades de pesquisa do método proposto. Algumas questões ainda precisam ser investigadas e estão listadas aqui como oportunidades de continuação.

(a) **Atualizações na carga de trabalho**. Discutiu-se na Seção 9.2 o desafio de selecionar ações de sintonia fina para cargas de trabalho com atualizações (inserções, atualizações, e exclusão). Não foi possível adaptar o modelo de custo proposto para considerar além do benefício das estruturas de acesso, penalidades relativas aos custos de manutenção. Entende-se que um cenário sem atualizações é possível, mas restringe a aplicação do método. Logo, faz-se necessário a consideração de atualizações por parte do modelo de custo.

(b) **Inclusão de novos especialistas**. Os três especialistas utilizados aqui trouxeram um bom desempenho para a carga de trabalho. Deseja-se explorar melhor o potencial de combinação com a extensão de outros especialistas, entre eles:

- *Índices clusterizados*. O especialista de índice aqui proposto gera apenas índices não-clusterizados, mas parece uma alternativa viável estudar o efeito da clusterização de VMAs por diferentes atributos. Talvez a combinação entre visões materializadas e índices clusterizados possa melhorar ainda mais o ganho destas estruturas de acesso.
- *Reescrita de consultas*. outra técnica passível de combinação pelo método aqui proposto seria reescrita de consultas. Araujo [69] apresenta um conjunto de onze heurísticas para reescrita de consultas SQL de forma automática. Um possível trabalho futuro seria modelar um especialista para gerar soluções combinadas também a partir destas heurísticas.
- *Particionamento horizontal*. Uma outra alternativa seria o particionamento horizontal como técnica passível de combinação com estruturas de acesso como índices e visões materializadas. No contexto do método proposto, as soluções combinadas poderiam sugerir particionamentos horizontais dos dados e as respectivas estruturas de acesso para as novas partições. Também seria uma possibilidade de integrar particionamento de dados e índices parciais, já que ambas técnicas exploram a identificação de faixas de valores dos dados para

particioná-los ou indexá-los e diminuir o custo de execução de consultas.

(c) **Ambientes virtualizados**. O método proposto foi planejado e avaliado para ambientes não-virtualizados tradicionais. Um trabalho futuro possível seria estudá-lo no contexto de arquiteturas virtualizadas e avaliar se os mecanismos de restrição de recursos computacionais também atenderiam às especificidades da virtualização. Talvez dentre as restrições usadas no método de combinação, poderia-se adicionar uma restrição monetária para ambientes onde o SGBD não é mantido pelo DBA, mas contratado como um serviço (*Software as a Service* - SaaS), ou quando os recursos computacionais são elásticos e podem ser adicionados ou reduzidos de acordo com a carga de trabalho (*Infrastructure as a Service - IaaS*). São cenários diferentes dos considerados neste trabalho, mas poderiam aumentar a abrangência de aplicação do método de combinação aqui proposto.

Referências bibliográficas

[1] R. Elmasri and S. Navathe, *Fundamentals of Database Systems*, 7th ed. Pearson Education, 2015.

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

[3] N. Bruno, *Automated Physical Database Design and Tuning*. CRC Press, 2012.

[4] R. Ramakrishnan and J. Gehrke, *Database Management Systems*, 3rd ed. New York, NY, USA: McGraw-Hill, Inc., 2008.

[5] M. A. V. Salles, "Criação autônoma de índices em bancos de dados," Master thesis, Pontifícia Universidade Católica do Rio de Janeiro - PUC-RIO, 2004.

[6] J. M. Monteiro, "Uma abordagem não-intrusiva para a manutenção automática do projeto físico de banco de dados," PhD thesis, Pontifícia Universidade Católica do Rio de Janeiro - PUC-RIO, 2008.

[7] J. M. Monteiro, A. Brayner, and S. Lifschitz, "Estado da Arte em Auto-Sintonia do Projeto Físico de BD. MCC35/08," *Monografias em Ciência da Computação - PUC-RIO*, 2008.

[8] M. Stonebraker, "The Case for Partial Indexes," *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, vol. 18, no. 4, pp. 4–11, 1989.

[9] R. Chirkova and J. Yang, "Materialized views," *Foundations and Trends in Databases*, vol. 4, no. 4, pp. 295–405, 2012.

[10] A. E. L. Lawler and D. E. Wood, "Branch-And-Bound Methods : A Survey Published," *Operations Research*, vol. 14, no. 4, pp. 699–719, 1966.

[11] O. Kwon, G. P. Im, and K. C. Lee, "An agent-based web service approach for supply chain collaboration," *Scientia Iranica*, vol. 18, no. 6, pp. 1545–1552, 2011.

[12] W. Brenner, H. Wittig, and R. Zarnekow, *Intelligent Software Agents: Foundations and Applications*. Springer-Verlag, 1998.

[13] S. Elfayoumy and J. Patel, "Database Performance Monitoring and Tuning Using Intelligent Agent Assistants," *Proceedings of the International Conference on Artificial Intelligence*, vol. 1, pp. 331—-335, 1999.

[14] E. M. T. Morelli, "Recriação Automática de Índices em um SGBD Relacional," Master thesis, Pontifícia Universidade Católica do Rio de Janeiro - PUC-RIO, 2006.

[15] F. A. C. A. Gonçalves, F. G. Guimarães, and M. J. F. Souza, "An Evolutionary Multi-agent System for Database Query Optimization," *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO)*, pp. 535–542, 2013.

[16] A. C. Almeida, A. Brayner, J. M. S. Monteiro, S. Lifschitz, and R. P. Oliveira, "Framework para Auto-Sintonia Fina Baseado em Planos Hipotéticos," *Anais do Simpósio Simpósio Brasileiro de Banco de Dados (SBBD)*, 2015.

[17] D. Mrozek, B. Malysiak-Mrozek, J. Mikolajczyk, and S. Kozielski, "Database Under Pressure – Testing Performance of Database Systems Using Universal Multi-Agent Platform," *Man-Machine Interactions 3*, pp. 631–641, 2014.

[18] M. F. Correa, "Modelos neuro-fuzzy hierárquicos com aprendizado por reforço para multi-agentes inteligentes," Ph.D. dissertation, Pontifícia Universidade Católica do Rio de Janeiro - PUC-RIO, Rio de Janeiro, Brazil, 2011.

[19] F. Zambonelli, N. R. Jennings, A. Omicini, and M. J. Wooldridge, "Agent-Oriented Software Engineering for Internet Applications," *Coordination of Internet Agents: Models, Technologies, and Applications*, pp. 326–346, 2001.

[20] S. Agrawal, S. Chaudhuri, and V. R. V. Narasayya, "Automated Selection of Materialized Views and Indexes in SQL Databases," *Proceedings of the International Conference on Very Large Data Bases (PVLDB)*, pp. 496–505, 2000.

[21] H. Kimura, G. Huo, A. Rasin, S. Madden, and S. B. Zdonik, "Coradd: Correlation aware database designer for materialized views and indexes,"

Proceedings of the International Conference on Very Large Data Bases (PVLDB), vol. 3, no. 1-2, pp. 1103–1113, 2010.

[22] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A K-Means clustering algorithm," *Applied Statistics*, vol. 28, no. 1, pp. 100–108, 1979.

[23] A. C. B. d. Almeida, "Framework para apoiar a sintonia fina de banco de dados," Ph.D. dissertation, Pontifícia Universidade Católica do Rio de Janeiro - PUC-RIO, 2013.

[24] R. P. d. Oliveira, E. Hermann, A. C. Almeida, and S. Lifschitz, "Projeto e implementação do framework Outer-tuning : auto sintonia e ontologia para bancos de dados relacionais," *Anais do Simpósio Brasileiro de Banco de Dados (SBBD)*, pp. 171–178, 2015.

[25] R. P. d. Oliveira and S. Lifschitz, "Sintonia fina baseada em ontologia : o caso de visões materializadas," *Anais do Workshop de teses e dissertações em banco de dados (WTDBD)*, pp. 346–352, 2014.

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

[27] T. Vijay Kumar and S. Kumar, "Materialized view selection using genetic algorithm," *Contemporary Computing*, vol. 306, pp. 225–237, 2012.

[28] K. Aouiche, P.-E. Jouve, and J. Darmont, "Clustering-based materialized view selection in data warehouses," *Proceedings of the East European Conference on Advances in Databases and Information Systems (ADBIS)*, pp. 81–95, 2006.

[29] G. Chan, Q. Li, and L. Feng, "Optimized design of materialized views in a real-life data warehousing environment," *International Journal of Information Technology*, vol. 7, no. 1, pp. 30–54, 2001.

[30] J. Phuboon-ob and R. Auepanwiriyakul, "Selecting Materialized Views Using Two-Phase Optimization with Multiple View Processing Plan," *Intl. Journal of Computer & Information Science & Engine*, pp. 166–171, 2007.

[31] T. Vijay Kumar and S. Kumar, "Materialized view selection using iterative improvement," *Advances in Computing and Information Technology*, vol. 178, pp. 205–213, 2013.

[32] R. Derakhshan and F. Dehne, "Simulated Annealing for Materialized View Selection in Data Warehousing Environment." *Proceedings of the International Conferences on Database and Applications (IASTED)*, pp. 89–94, 2006.

[33] X. Sun and Z. Wang, "An Efficient Materialized Views Selection Algorithm Based on PSO," *International Intelligent Systems and Applications (IJISA)*, pp. 1–4, 2009.

[34] Z. Yuhang, L. Qi, and Y. Wei, "Materialized view selection algorithm cssa vsp," *Proceedings of Computational Intelligence and Natural Computing (CINC)*, vol. 1, pp. 68–71, 2010.

[35] X. Li, X. Qian, J. Jiang, and Z. Wang, "Shuffled Frog Leaping Algorithm for Materialized Views Selection," *Workshop Education Technology and Computer Science (ICETC)*, pp. 7–10, 2010.

[36] M. Lawrence, "Multiobjective genetic algorithms for materialized view selection in OLAP data warehouses," *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO)*, pp. 1231–1238, 2006.

[37] S. Talebian and S. Abdul Kareem, "Using genetic algorithm to select materialized views subject to dual constraints," *Proceedings of International Conference on Signal Processing Systems (ICSPS)*, pp. 633–638, 2009.

[38] C. Maier, D. Dash, I. Alagiannis, A. Ailamaki, and T. Heinis, "PARINDA: An Interactive Physical Designer for PostgreSQL," *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pp. 701–704, 2010.

[39] E. Barcucci, R. Pinzani, and R. Sprugnoli, "Optimal Selection of Secondary Indexes," *IEEE Transactions on Software Engineering*, vol. 16, no. 1, pp. 32–38, 1990.

[40] A. Caprara, M. Fischetti, and D. Maio, "Exact and approximate algorithms for the index selection problem in physical database design," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 7, no. 6, pp. 955–967, 2002.

[41] M. R. Frank, E. R. Omiecinski, and S. B. Navathe, "Adaptive and automated index selection in RDBMS," *Proceedings of the International*

Conference on Extending Database Technology (EDBT), vol. 580, pp. 277–292, 1992.

[42] M. Zaman, J. Surabattula, and L. Gruenwald, "An auto-indexing technique for databases based on clustering," *Proceedings of the International Conference on Database and Expert Systems Applications (DEXA)*, pp. 776–780, 2004.

[43] S. Chaudhuri and V. Narasayya, "Microsoft index turning wizard for SQL Server 7.0," *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, vol. 27, no. 2, pp. 553–554, 1998.

[44] S. Chaudhuri, M. Datar, and V. Narasayya, "Index selection for databases: A hardness study and a principled heuristic solution," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 16, no. 11, pp. 1313–1323, 2004.

[45] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," *Journal of Computer Science and Technology*, vol. 15, no. 6, pp. 487–499, 1994.

[46] G. Valentin, M. Zuliani, D. Zilio, G. Lohman, and A. Skelley, "DB2 advisor: an optimizer smart enough to recommend its own indexes," *Proceedings of IEEE International Conference on Data Engineering (ICDE)*, pp. 101–110, 2000.

[47] P. Seshadri and A. Swami, "Generalized partial indexes," *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pp. 420–427, 1995.

[48] P. Cudre-Mauroux, E. Wu, and S. Madden, "Trajstore: An adaptive storage system for very large trajectory data sets," *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pp. 109–120, 2010.

[49] K.-L. Tan, S. Wu, L. Shou, and P. Lu, "An efficient and compact indexing scheme for large-scale data store," *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pp. 326–337, 2013.

[50] S. Idreos, M. L. Kersten, and S. Manegold, "Updating a cracked database," *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pp. 413–424, 2007.

[51] M. Kersten, S. Manegold, and S. J. Amsterdam, "Cracking the Database Store," *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, pp. 213–224, 2005.

[52] A. Domínguez and S. Lifschitz, "Database self-tuning with partial indexes," Master Thesis, Pontifícia Universidade Católica do Rio de Janeiro - PUC-RIO, 2016.

[53] C. Zheng, Z. Ding, and J. Hu, "Self-tuning performance of database systems with neural network," *Lecture Notes in Computer Science (LNCS)*, vol. 8588, pp. 1–12, 2014.

[54] D. V. Aken, A. Pavlo, G. J. Gordon, and B. Zhang, "Automatic database management system tuning through large-scale machine learning," *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pp. 1009–1024, 2017.

[55] B. Zhang, D. Van Aken, J. Wang, T. Dai, S. Jiang, J. Lao, S. Sheng, A. Pavlo, and G. J. Gordon, "A demonstration of the ottertune automatic database management system tuning service," *Proceedings of the International Conference on Very Large Data Bases (PVLDB)*, vol. 11, no. 12, pp. 1910–1913, 2018.

[56] J. Zhang, L. Liu, M. Ran, Z. Li, Y. Liu, K. Zhou, G. Li, Z. Xiao, B. Cheng, J. Xing, Y. Wang, and T. Cheng, "An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning," *Proceedings of the International Conference on Management of Data (SIGMOD)*, pp. 415–432, 2019.

[57] Oracle Corporate, "Automatic SQL Tuning," 2019. [Online]. Available: https://docs.oracle.com/cd/B19306_01/server.102/b14211/sql_tune.htm#g42443

[58] Oracle Group, "Autonomous Database," 2019. [Online]. Available: https://www.oracle.com/database/what-is-autonomous-database.html

[59] Microsoft Company, "SQL Server 2017 Automatic Tuning," 2017. [Online]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/automatic-tuning/automatic-tuning?view=sql-server-2017

[60] Ankane, "Dexter Automatic Index Generator," 2019. [Online]. Available: https://github.com/ankane/dexter

[61] J. Rouhaud, "HypoPG - hypothetical indexes for PostgreSQL," 2019. [Online]. Available: https://hypopg.readthedocs.io/en/latest/

[62] PoWA-team, "PostgreSQL Workload Analyzer - POWA," 2019. [Online]. Available: https://powa.readthedocs.io/en/latest/

[63] P. Horn, "Autonomic Computing: IBM's Perspective on the State of Information Technology," *International Business Machines*, 2001.

[64] K.-U. Sattler, I. Geist, and E. Schallehn, "QUIET: Continuous Query-driven Index Tuning," *Proceedings of the International Conference on Very Large Data Bases (PVLDB)*, pp. 1129–1132, 2003.

[65] J. Magnusson and D. D. Dankel, "Proactive database index tuning through data threshold prediction," *Proceedings of the ACM Annual Southeast Regional Conference*, p. 1, 2009.

[66] L. C. Castro, "Sintonia fina de sistemas de gerenciamento de banco de dados em ambientes virtualizados," Master Thesis, Pontifícia Universidade Católica do Rio de Janeiro - PUC-RIO, Rio de Janeiro, Brazil, 2017.

[67] D. Tsirogiannis, S. Harizopoulos, M. A. Shah, J. L. Wiener, and G. Graefe, "Query processing techniques for solid state drives," *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, p. 59, 2009.

[68] S. Chaudhuri and V. Narasayya, "Self-tuning database systems: A decade of progress," *Proceedings of the International Conference on Very Large Data Bases (PVLDB)*, pp. 3–14, 2007.

[69] A. H. M. de Araújo, J. M. Monteiro, and J. A. F. de Macêdo, "Uma Abordagem Não-Intrusiva para Sintonia Automática de Consultas SQL," *X Workshop de Teses e Dissertações em Banco de Dados, WTBD*, 2010.

[70] S. H. Talebian and S. A. Kareem, "A lexicographic ordering genetic algorithm for solving multi-objective view selection problem," *Proceedings of the International Conference on Computer Research and Development (ICCRD)*, pp. 110–115, 2010.

[71] T. V. Vijay Kumar, M. Haider, and S. Kumar, "Proposing Candidate Views for Materialization," *Information Systems, Technology and Management*, vol. 54, pp. 89–98, 2010.

[72] T. Vijay Kumar and A. Ghoshal, "A reduced lattice greedy algorithm for selecting materialized views," *Information Systems, Technology and Management*, vol. 31, pp. 6–18, 2009.

[73] E. Baralis, S. Paraboschi, and E. Teniente, "Materialized views selection in a multidimensional database," *Proceedings of the International Conference on Very Large Data Bases (PVLDB)*, pp. 156–165, 1997.

[74] K. Schnaitter, S. Abiteboul, T. Milo, and N. Polyzotis, "COLT: Continuous On-line Tuning," *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pp. 1–23, 2006.

[75] L. Bellatreche, K. Boukhalfa, and M. Mohania, "Pruning Search Space of Physical Database Design," *Proceedings of the International Conference on Database and Expert Systems Applications (DEXA)*, vol. 63, no. 8, pp. 479–488, 2013.

[76] S. Chaudhuri and G. Weikum, "Foundations of automated database tuning," *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pp. 964–965, 2006.

[77] S. Chen, M. A. Nascimento, B. C. Ooi, and K.-L. Tan, "Continuous online index tuning in moving object databases," *ACM Transactions on Database Systems (TODS)*, vol. 35, no. 3, pp. 1–51, 2010.

[78] Q. T. Tran, I. Jimenez, R. Wang, N. Polyzotis, and A. Ailamaki, "Rita: An index-tuning advisor for replicated databases," *Proceedings of the Conference on Scientific and Statistical Database Management (SSDBM)*, pp. 22:1–22:12, 2015.

[79] A. Y. M. Barrientos, "Uma arquitetura para auto-sintonia global de SGBDs usando agentes," Master Thesis, Pontifícia Universidade Católica do Rio de Janeiro - PUC-RIO, Rio de Janeiro, Brazil, 2004.

[80] A. W. Carvalho, "Criação Automática de Visões Materializadas em SGBDs Relacionais," Master Thesis, Pontifícia Universidade Católica do Rio de Janeiro - PUC-RIO, 2011.

[81] A. S. d. Medeiros, "Particionamento como ação de sintonia fina em bancos de dados relacionais," Master Thesis, Pontifícia Universidade Católica do Rio de Janeiro - PUC-RIO, Rio de Janeiro, Brazil, apr 2017.

[82] R. L. d. C. Costa, S. Lifschitz, and M. A. V. Salles, "Index self-tuning with agent-based databases," *Proceedings of the Latin-American Conference on Informatics (CLEI)*, vol. 6, pp. 1–22, 2003.

[83] H. S. Gupta, "Selection and Maintenance of Views in a Data Warehouse," Ph.D. dissertation, Stanford University, Stanford, CA, USA, 2000.

[84] B. Hayes-Roth, "A blackboard architecture for control," *Artificial Intelligence*, vol. 26, no. 3, pp. 251–321, 1985.

[85] L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D. R. Reddy, "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty," *ACM Computing Surveys*, vol. 12, no. 2, pp. 213–253, 1980.

[86] The PostgreSQL Global Development Group, "Explain PostgreSQL," 2019. [Online]. Available: https://www.postgresql.org/docs/9.6/sql-explain.html

[87] D. Pisinger, "Where are the hard knapsack problems?" *Computers and Operations Research*, vol. 32, no. 9, pp. 2271–2284, 2005.

[88] S. Martello, D. Pisinger, and P. Toth, "Dynamic Programming and Strong Bounds for the 0-1 Knapsack Problem," *Manage. Sci.*, vol. 45, no. 3, pp. 414–424, mar 1999.

[89] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., 1990.

[90] A. Y. Milanés and S. Lifschitz, "Design and Implementation of a Global Self-tuning Architecture," *Anais do Simpósio Brasileiro de Banco de Dados (SBBD)*, pp. 70–84, 2005.

[91] R. d. C. Costa, S. Lifschitz, M. de Noronha, and M. Vaz Salles, "Implementation of an agent architecture for automated index tuning," *21st International Conference on Data Engineering Workshops*, pp. 1215–1215, April 2005.

[92] G. Di Marzo Serugendo, M. P. Gleizes, and A. Karageorgos, *Self-organising Software: From Natural to Artificial Adaptation*. Natural Computing Series, Springer, 2011.

[93] M. Beth, "a Self-Organizing Systems Perspective on Planning for Sustainability," Master Thesis, University of Waterloo, 1998.

[94] P. J. Gmytrasiewicz and E. H. Durfee, "Decision-theoretic recursive modeling and the coordinated attack problem," *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, pp. 88–95, 1992.

[95] SpA, Telecom Italia, "JADE," 2019. [Online]. Available: http://jade.tilab.com/

[96] IEEE, Computer Society, "Foundation for Inteligent Physical Agents," 2019.

[97] Microsoft Corporate, "SQL Server 2017 Developer," 2019. [Online]. Available: https://www.microsoft.com/pt-br/sql-server/sql-server-2017-editions

[98] Oracle Corporate, "Oracle 12c," 2019. [Online]. Available: https://www.oracle.com/corporate/features/database-12c/

[99] The PostgreSQL Global Development Group, "PostgreSQL 9.7," 2019. [Online]. Available: https://www.postgresql.org/docs/9.7

[100] J. M. Monteiro, S. Lifschitz, and A. Brayner, "Extraindo Metadados de SGBDs," *Monografias Técnicas em Informática (MCC) - PUC-Rio*, no. 07, 2007.

[101] Transaction Processing Performance Council, "TPC," 2019. [Online]. Available: http://www.tpc.org/

[102] T. P. P. Council, "TPC-H," 2019. [Online]. Available: http://www.tpc.org/tpch/

[103] A. A. Vaisman, A. O. Mendelzon, W. Ruaro, and S. G. Cymerman, "Supporting dimension updates in an OLAP server," *Information Systems*, vol. 29, no. 2, pp. 165–185, 2004.

[104] S. Agrawal, E. Chu, and V. Narasayya, "Automatic physical design tuning: Workload as a sequence," *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pp. 683–694, 2006.

[105] N. Bruno and S. Chaudhuri, "An Online Approach to Physical Design Tuning," *2007 IEEE 23rd International Conference on Data Engineering*, pp. 826–835, apr 2007.

[106] I. Mami and Z. Bellahsene, "A survey of view selection methods," *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, vol. 41, no. 1, pp. 20–29, Apr. 2012.

A
Modelos de consulta do TPC-H

```
/* TPC_H Query 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 '?' - interval '90' day
GROUP BY
    l_returnflag,
    l_linestatus
ORDER BY
    l_returnflag,
    l_linestatus;
```

```

/* TPC_H Query 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 = p_p_size
  AND P_TYPE LIKE '%%?'
  AND S_NATIONKEY = N_NATIONKEY
  AND N_REGIONKEY = R_REGIONKEY
  AND R_NAME = '?'
  AND PS_SUPPLYCOST =
    (SELECT MIN(PS_SUPPLYCOST) AS MIN_SUPPLY
     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 = '?')
ORDER BY S_ACCTBAL DESC,
         N_NAME,
         S_NAME,
         P_PARTKEY
limit 100;
```

```

/* TPC_H Query 3 */
SELECT
    l_orderkey,
    sum(l_extendedprice * (1 - l_discount)) as revenue,
    o_orderdate,
    o_shippriority
FROM
    customer,
    orders,
    lineitem
WHERE
    c_mktsegment = '?'
    AND c_custkey = o_custkey
    AND l_orderkey = o_orderkey
    AND o_orderdate < date '?'
    AND l_shipdate > date '?'
GROUP BY
    l_orderkey,
    o_orderdate,
    o_shippriority
ORDER BY
```

```
    revenue desc,
    o_orderdate
LIMIT 20;
```

```

/* TPC_H Query 4 */
select
        o_orderpriority,
        count( * ) as order_count
from
        orders
where
        o_orderdate >= date '?'
        and o_orderdate < date '?' + interval '?'
        and exists (
                select
                        *
                from
                        lineitem
                where
                        l_orderkey = o_orderkey
                        and l_commitdate < l_receiptdate
        )
group by
        o_orderpriority
order by
        o_orderpriority;
```

```

/* TPC_H Query 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 = 'p_r_name'
```

```
    AND o_orderdate >= date '?'
    AND o_orderdate < date '?' + interval '1' year
GROUP BY
    n_name
ORDER BY
    revenue desc;
```

```
/* TPC_H Query 6 */
SELECT
    sum(l_extendedprice * l_discount) as revenue
FROM
    lineitem
WHERE
    l_shipdate >= date '?'
    AND l_shipdate < date '?' + interval '1' year
    AND l_discount between ?-1 AND ?+1
    AND l_quantity < ?;
```

```
/* TPC_H Query 7 */
SELECT NA_N_NAME,
       NB_N_NAME,
       L_YEAR,
       SUM(VOLUME) AS REVENUE
FROM
  (SELECT NA.N_NAME AS NA_N_NAME,
          NB.N_NAME AS NB_N_NAME,
          date_part('year', L_SHIPDATE) AS L_YEAR,
          L_EXTENDEDPRICE*(1-L_DISCOUNT) AS VOLUME
   FROM SUPPLIER,
        LINEITEM,
        ORDERS,
        CUSTOMER,
        NATION NA,
        NATION NB
   WHERE S_SUPPKEY = L_SUPPKEY
     AND O_ORDERKEY = L_ORDERKEY
     AND C_CUSTKEY = O_CUSTKEY
     AND S_NATIONKEY = NA.N_NATIONKEY
     AND C_NATIONKEY = NB.N_NATIONKEY
     AND ((NA.N_NAME = '?'
           AND NB.N_NAME = '?')
          OR (NA.N_NAME = '?'
              AND NB.N_NAME = '?'))
     AND L_SHIPDATE BETWEEN '?' AND date '?' + interval '1
        ' year ) AS SHIPPING
GROUP BY NA_N_NAME,
         NB_N_NAME,
         L_YEAR
```

```
ORDER BY NA_N_NAME,
         NB_N_NAME,
         L_YEAR;
```

```
/* TPC_H Query 8 */
SELECT O_YEAR,
       SUM(CASE
               WHEN NB_N_NAME = 'BRAZIL' THEN VOLUME
               ELSE 0
           END)/SUM(VOLUME) AS MKT_SHARE
FROM
  (SELECT date_part('year', O_ORDERDATE) AS O_YEAR,
          L_EXTENDEDPRICE*(1-L_DISCOUNT) AS VOLUME,
          NB.N_NAME AS NB_N_NAME
   FROM PART,
        SUPPLIER,
        LINEITEM,
        ORDERS,
        CUSTOMER,
        NATION NA,
        NATION NB,
        REGION
   WHERE P_PARTKEY = L_PARTKEY
     AND S_SUPPKEY = L_SUPPKEY
     AND L_ORDERKEY = O_ORDERKEY
     AND O_CUSTKEY = C_CUSTKEY
     AND C_NATIONKEY = NA.N_NATIONKEY
     AND NA.N_REGIONKEY = R_REGIONKEY
     AND R_NAME = ':'
     AND S_NATIONKEY = NB.N_NATIONKEY
     AND O_ORDERDATE BETWEEN '?' AND date '?' + interval '
        1' year
     AND P_TYPE= '?') AS ALL_NATIONS
GROUP BY O_YEAR
ORDER BY O_YEAR;
```

```

/* TPC_H Query 9 */
SELECT NATION,
       O_YEAR,
       SUM(AMOUNT) AS SUM_PROFIT
FROM
  (SELECT N_NAME AS NATION,
          date_part('year', 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 '%%?%%') AS PROFIT
GROUP BY NATION,
         O_YEAR
ORDER BY NATION,
         O_YEAR DESC;
```

```
/* TPC_H Query 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 '?' AND
        O_ORDERDATE < date '?' + interval '3' month AND
        L_RETURNFLAG = '?' 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
LIMIT 20;
```

```
/* TPC_H Query 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 = '?'
GROUP BY PS_PARTKEY
HAVING SUM(PS_SUPPLYCOST*PS_AVAILQTY) >
  (SELECT SUM(PS_SUPPLYCOST*PS_AVAILQTY) * 0.0001000000 as
      cost_sum
   FROM PARTSUPP,
        SUPPLIER,
        NATION
   WHERE PS_SUPPKEY = S_SUPPKEY
     AND S_NATIONKEY = N_NATIONKEY
     AND N_NAME = '?')
ORDER BY VALUE DESC;
```

```
/* TPC_H Query 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 <> '3-medium'
            AND o_orderpriority <> '4-not␣specified'
            then 1
        else 0
    end) AS low_line_count
FROM
    orders,
    lineitem
WHERE
    o_orderkey = l_orderkey
    AND l_shipmode in ('?', '?')
    AND l_commitdate < l_receiptdate
    AND l_shipdate < l_commitdate
    AND l_receiptdate >= date '?'
```

```
    AND l_receiptdate < date '?' + interval '1' year
GROUP BY
    l_shipmode
ORDER BY
    l_shipmode;
```

```
/* TPC_H Query 13 */
SELECT C_COUNT,
       COUNT( * ) AS CUSTDIST
FROM
  (SELECT C_CUSTKEY,
          COUNT(O_ORDERKEY) as count_orderkey
   FROM CUSTOMER
   LEFT OUTER JOIN ORDERS ON C_CUSTKEY = O_CUSTKEY
   AND O_COMMENT NOT LIKE '%%?%%'
   GROUP BY C_CUSTKEY) AS C_ORDERS (C_CUSTKEY, C_COUNT)
GROUP BY C_COUNT
ORDER BY CUSTDIST DESC,
         C_COUNT DESC;
```

```
/* TPC_H Query 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
WHERE
    l_partkey = p_partkey
    AND l_shipdate >= date '?'
    AND l_shipdate < date '?' + interval '1' month;
```

```
/* TPC_H Query 15 */
SELECT l_suppkey,
       sum(l_extendedprice * (1 - l_discount)) as
          sum_price
FROM lineitem
WHERE l_shipdate >= date '?'
  AND l_shipdate < date '?' + interval '3' MONTH
GROUP BY l_suppkey;
```

```
/* TPC_H Query 16 */
select p_brand,
       p_type,
```

```
       p_size,
       count(distinct(ps_suppkey)) as supp_cnt
from partsupp,
     part
where p_partkey = ps_partkey
  and p_brand <> '?'
  and p_type ilike '%?%'
  and p_size in (p_p_size,
                 p_p_size,
                 p_p_size,
                 p_p_size,
                 p_p_size,
                 p_p_size,
                 p_p_size,
                 p_p_size)
  and ps_suppkey not in
    (select s_suppkey
     from supplier
     where s_comment like '%%?%%')
group by p_brand,
         p_type,
         p_size
order by supp_cnt desc,
         p_brand,
         p_type,
         p_size;
```

```
/* TPC_H Query 17 */
SELECT SUM(L_EXTENDEDPRICE)/7.0 AS AVG_YEARLY
FROM LINEITEM,
     PART
WHERE P_PARTKEY = L_PARTKEY
  AND P_BRAND = '?'
  AND P_CONTAINER = '?'
  AND L_QUANTITY <
    (SELECT 0.2*AVG(L_QUANTITY) as avg_quantity
     FROM LINEITEM
     WHERE L_PARTKEY = P_PARTKEY);
```

```
/* TPC_H Query 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) > ?)
  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
LIMIT 100;
```

```
/* TPC_H Query 19 */
select
        sum(l_extendedprice* (1 - l_discount)) as revenue
from
        lineitem,
        part
where
        (
                p_partkey = l_partkey
                and p_brand = '?'
                and p_container in ('?', '?', '?', '?')
                and l_quantity >= ? and l_quantity <= ? +
                    10
                and p_size between ? and ? + 15
                and l_shipmode in ('?', '?')
                and l_shipinstruct = '?'
        )
        or
        (
                p_partkey = l_partkey
                and p_brand = '?'
                and p_container in ('?', '?', '?', '?')
                and l_quantity >= ? and l_quantity <= ? +
                    10
                and p_size between ? and ? + 15
                and l_shipmode in ('?', '?')
                and l_shipinstruct = '?'
        )
        or
        (
```

```
                p_partkey = l_partkey
                and p_brand = '?'
                and p_container in ('?', '?', '?', '?')
                and l_quantity >= ? and l_quantity <= ? +
                    10
                and p_size between ? and ? + 15
                and l_shipmode in ('?', '?')
                and l_shipinstruct = '?'
        );
```

```
/* TPC_H Query 20 */
SELECT S_NAME,
       S_ADDRESS
FROM SUPPLIER,
     NATION
WHERE S_SUPPKEY IN
    (SELECT PS_SUPPKEY
     FROM PARTSUPP
     WHERE PS_PARTKEY IN
         (SELECT P_PARTKEY
          FROM PART
          WHERE P_NAME LIKE '%?%')
       AND PS_AVAILQTY >
         (SELECT 0.5*sum(L_QUANTITY) as sum_quantity
          FROM LINEITEM
          WHERE L_PARTKEY = PS_PARTKEY
            AND L_SUPPKEY = PS_SUPPKEY
            AND L_SHIPDATE >= '?'
            AND L_SHIPDATE < date '?' + interval '1' year
                )
        )
  AND S_NATIONKEY = N_NATIONKEY
  AND N_NAME = '?'
ORDER BY S_NAME;
```

```
/* TPC_H Query 21 */
SELECT S_NAME,
           COUNT( * ) AS NUMWAIT
FROM SUPPLIER,
     LINEITEM LA,
     ORDERS,
     NATION
WHERE S_SUPPKEY = LA.L_SUPPKEY
  AND O_ORDERKEY = LA.L_ORDERKEY
  AND O_ORDERSTATUS = '?'
  AND LA.L_RECEIPTDATE> LA.L_COMMITDATE
  AND EXISTS
    (SELECT *
```

```
      FROM LINEITEM LB
      WHERE LB.L_ORDERKEY = LA.L_ORDERKEY
        AND LB.L_SUPPKEY <> LA.L_SUPPKEY)
  AND NOT EXISTS
    (SELECT *
      FROM LINEITEM LC
      WHERE LC.L_ORDERKEY = LA.L_ORDERKEY
        AND LC.L_SUPPKEY <> LA.L_SUPPKEY
        AND LC.L_RECEIPTDATE > LC.L_COMMITDATE)
  AND S_NATIONKEY = N_NATIONKEY
  AND N_NAME = '?'
GROUP BY S_NAME
ORDER BY NUMWAIT DESC,
         S_NAME
LIMIT 100;
```

```
/* TPC_H Query 22 */
select
        cntrycode,
        count( * ) numcust,
        sum(c_acctbal) totacctbal
from
        (
                select
                        substr(c_phone , 0, 3) cntrycode,
                        c_acctbal
                from
                        customer
                where
                        substr(c_phone , 0, 3) in
                                ('?', '?', '?', '?', '?',
                                   '?', '?')
                        and c_acctbal > (
                                select
                                        avg(c_acctbal)
                                           avg_acctbal
                                from
                                        customer
                                where
                                        c_acctbal > 0.00
                                        and substr(c_phone
                                             , 0, 3) in
                                            ('?', '?',
                                                '?', '
                                               ?', '?'
                                               , '?',
                                               '?')
```

```
25                          )
26                          and not exists (
27                                  select
28                                          *
29                                  from
30                                          orders
31                                  where
32                                          o_custkey =
                                               c_custkey
33                          )
34          ) custsale
35  group by
36          cntrycode
37  order by
38          cntrycode;
```