



Victor Augusto Lima Lins de Souza

**TuningChef: uma abordagem para escolher as
ações de sintonia fina de banco de dados com
melhor custo-benefício**

Dissertação de Mestrado

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

Orientador: Prof. Sérgio Lifschitz

Rio de Janeiro
Setembro de 2022



Victor Augusto Lima Lins de Souza

**TuningChef: uma abordagem para escolher as
ações de sintonia fina de banco de dados com
melhor custo-benefício**

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre 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^a. Fernanda Araujo Baião Amorim

Departamento de Engenharia Industrial – PUC-Rio

Prof^a. Juliana Alves Pereira

Departamento de Informática – PUC-Rio

Prof. Javam de Castro Machado

UFC

Rio de Janeiro, 15 de Setembro de 2022

Todos os direitos reservados. A reprodução, total ou parcial do trabalho, é proibida sem a autorização da universidade, do autor e do orientador.

Victor Augusto Lima Lins de Souza

Possui graduação em Engenharia da Computação (2018) pela Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio). Atualmente atua como líder técnico na Stone Pagamentos. Tem experiência nas áreas de engenharia de software, arquitetura de sistemas e bancos de dados relacionais.

Ficha Catalográfica

Souza, Victor Augusto Lima Lins de

TuningChef: uma abordagem para escolher as ações de sintonia fina de banco de dados com melhor custo-benefício / Victor Augusto Lima Lins de Souza; orientador: Sérgio Lifschitz. – 2022.

74 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2022.

Inclui bibliografia

1. Engenharia da Computação – Teses. 2. Banco de dados. 3. Sintonia fina. 4. Tomada de decisão. 5. Visões materializadas. 6. Índices. 7. Estrutura hipotéticas. I. Lifschitz, Sérgio. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

To my parents, for their support
and encouragement.

Agradecimentos

Gostaria de agradecer aos meus pais, à minha irmã, ao meu orientador, à minha namorada, aos meus amigos, e aos colegas da PUC-Rio, por todo o apoio no decorrer dessa jornada.

Agradeço aos meus pais, Augusto e Ana Paula, e a minha irmã, Karina, que sempre buscaram me motivar e me fazer entender o valor de meu trabalho, me dando forças para ir até o final e conseguir o título de mestre, mesmo nos momentos onde eu constantemente me questionava.

Agradeço ao meu orientador, Professor Sérgio Lifschitz, por, em um momento tão complicado para se orientar devido a pandemia, ter me dado todo o suporte para conseguir equilibrar o trabalho e os estudos, fornecendo toda a orientação e motivação necessária para chegar no final dessa jornada.

Agradeço a minha namorada, Mariana, e aos meus amigos, não só pelo apoio mas pela compreensão dos dias, que se tornaram semanas, nos quais eu era ausente, mas que mesmo assim nunca saíram do meu lado.

Agradeço a banca por terem disponibilizado tempo necessário para avaliarem trabalho, apresentando críticas fundamentais para a versão final deste trabalho.

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

Resumo

Souza, Victor Augusto Lima Lins de; Lifschitz, Sérgio. **TuningChef: uma abordagem para escolher as ações de sintonia fina de banco de dados com melhor custo-benefício**. Rio de Janeiro, 2022. 74p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Enquanto muitos trabalhos de pesquisa propõem uma forma de listar um conjunto de opções de sintonia fina para uma determinada carga de trabalho, poucos oferecem uma maneira de ajudar o DBA a tomar melhores decisões ao encontrar um conjunto de ações disponíveis. TuningChef é o resultado do desenvolvimento de uma proposta do passo a passo desse processo de decisão. Dado um conjunto de opções de sintonia fina, recomendamos um subconjunto com boa proporção de custo-benefício, com contexto suficiente para que o DBA entenda a motivação por trás de cada decisão, incluindo a possibilidade de deixar o usuário construir seu próprio subconjunto e verificar o impacto esperado. Também são apresentados resultados experimentais que demonstram a importância do processo de decisão, onde dentro de um subconjunto de 50+ ações de sintonia fina sugeridas por uma ferramenta externa, apenas 8 mostram-se como benéficas para a carga de trabalho utilizada.

Palavras-chave

Banco de dados; Sintonia fina; Tomada de decisão; Visões materializadas; Índices; Estrutura hipotéticas.

Abstract

Souza, Victor Augusto Lima Lins de; Lifschitz, Sérgio (Advisor).
TuningChef: an approach for choosing the best cost-benefit database tuning actions. Rio de Janeiro, 2022. 74p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

While many research works propose a way to list a set of fine-tuning options for a given workload, only a few offer a way to help the DBA make better decisions when encountering a set of available options, especially when taking his possibilities into consideration. We propose and develop a step-by-step decision process. Given a set of fine-tuning options, we recommend a subset with good cost-benefit proportion. Enough context for the DBA accompanies the recommendation to understand its reasoning, with the possibility of letting the user build his own subset and check the expected impact. Some experimental results are also described, showing the importance of the decision step when fine tuning a database, where in a set on 50+ fine tuning actions suggested by an external tool, only 8 are considered beneficial for the a specific workload.

Keywords

Database systems; Tuning; Decision taking; Materialized views; Indexes; Hypothetical structures.

Sumário

1	Introdução	13
1.1	Objetivos e escopo da dissertação	13
1.2	Motivação	14
1.3	Definição do Problema	15
1.4	Estrutura da dissertação	15
2	Conceitos Básicos e Trabalhos Relacionados	16
2.1	Carga de Trabalho	16
2.2	Sintonia Fina	16
2.3	Índices	17
2.3.1	Índices Parciais	18
2.3.2	Índices Compostos	18
2.4	Visões	19
2.5	Heurística de Benefícios	19
2.6	Contexto e Trabalhos Relacionados	20
2.7	Resumo do Capítulo	23
3	Proposta de solução	24
3.1	Motivação	24
3.2	Requisitos funcionais	26
3.3	Projeto de arquitetura da TuningChef	27
3.4	Conclusão	31
4	Desenvolvimento da TuningChef	32
4.1	Tecnologias utilizadas	32
4.2	Coletor de Carga	32
4.2.1	Arquivo de entrada	33
4.2.2	Formatos de saída	33
4.2.3	Algoritmo de coleta	34
4.2.4	PostgreSQL	35
4.2.5	MySQL	37
4.3	Avaliador	38
4.3.1	PostgreSQL	41
4.3.1.1	Índices Hipotéticos	41
4.3.1.2	Visões Materializadas Hipotéticas	42
4.3.2	SQLServer	44
4.3.2.1	Índices Hipotéticos	44
4.4	Exemplo de execução do Algoritmo SASF	46
4.5	Interface	48
4.6	Conclusão	52
5	Resultados experimentais	53
5.1	Ambiente de teste	53
5.2	Limitações	54

5.3	Análise dos resultados	54
5.4	Conclusão	58
6	Conclusão	59
6.1	Contribuições da pesquisa	59
6.2	Trabalhos futuros	60
	Referências bibliográficas	62
A	Consultas TPC-H	66
B	Ações de sintonia fina	71

Lista de figuras

Figura 3.1	Exemplo de um processo de sintonia fina	24
Figura 3.2	<i>Arquitetura Outer-Tuning</i>	28
Figura 3.3	Arquitetura TuningChef	30
Figura 4.1	Exemplo de execução do Algoritmo 2	46
Figura 4.2	Tela inicial da <i>TuningChef</i>	48
Figura 4.3	Consultas presentes na carga de trabalho	50
Figura 4.4	Detalhamento da ação	51
Figura 4.5	Ações escolhidas	51
Figura 4.6	Consultas atualizadas	52
Figura 5.1	Ações de sintonia fina disponíveis	54
Figura 5.2	Peso das consultas	55
Figura 5.3	Ações de sintonia fina recomendadas aplicadas	56
Figura 5.4	Consultas após recomendação aplicada	56
Figura 5.5	Resultado da intervenção manual	57
Figura A.1	Consulta TPC-H Q1.1	66
Figura A.2	Consulta TPC-H Q1.2	66
Figura A.3	Consulta TPC-H Q1.3	67
Figura A.4	Consulta TPC-H Q3.1	67
Figura A.5	Consulta TPC-H Q3.2	68
Figura A.6	Consulta TPC-H Q4	68
Figura A.7	Consulta TPC-H Q5	69
Figura A.8	Consulta TPC-H Q8	69
Figura B.1	Índices Simples	71
Figura B.2	Índices Compostos	72
Figura B.3	Índices Parciais	73
Figura B.4	Visão Materializada para as consultas Q3	74

Lista de Abreviaturas

BnB – *Branch-and-bound*

CTE – *Common Table Expression*

DBA – *Database Administrator*

RL – *Reinforcement Learning*

SGBD – Sistema Gerenciador de Banco de Dados

TPS – Transações por segundo

My beautifull epigraph

Wassily Kandinsky, *Regards sur le passé.*

1

Introdução

O processo de sintonia fina em banco de dados relacionais é um assunto muito debatido na academia, com trabalhos frequentemente publicados e evoluções constantes. Entretanto uma das etapas do processo ainda é muito pouco explorada: a decisão de quais ações de sintonia fina devem ser executadas no banco. Quando um administrador de banco de dados (DBA) se depara com um conjunto de múltiplas ações de sintonia fina, o mesmo precisa escolher, entre elas, as melhores ações disponíveis de forma que elas tenham um bom custo-benefício para sua carga de trabalho.

Neste trabalho vamos explorar essa etapa ainda pouco formal e, de certa forma, considerada como artística por ser pouco sistematizada e muito dependente da experiência do administrador do banco de dados. Serão apresentadas motivações por trás desse estudo e uma proposta de solução no formato de uma nova arquitetura, viabilizada através da implementação de uma ferramenta chamada *TuningChef*. A solução tem como principal objetivo apoiar o processo de decisão do DBA, levando seus conhecimentos em consideração. Para isso o DBA pode interagir com a ferramenta, se desejar, para alterar as opções recomendadas pela *TuningChef*.

Essa pesquisa é parte de um processo que busca completar um ciclo de mais de uma década do grupo de pesquisa BioBD da PUC-Rio, que possui diversos trabalhos publicados na área.

1.1

Objetivos e escopo da dissertação

Esse trabalho tem foco na etapa de decisão do processo de *tuning* de um banco de dados relacional, de forma a propor uma solução para apoiar o DBA neste momento de decisão. Esta etapa é pouco formal e pouco explorada na academia e sua sistematização é um desafio. Para que isso seja feito, foi necessário:

- mapear e coletar as informações necessárias de uma carga de trabalho
- colocar o conhecimento do DBA em um algoritmo
- desenvolver uma solução para análise de custo e benefício de possíveis ações de tuning

- desenvolver uma interface para uso do DBA

Nesta dissertação, assim como em outras pesquisas do grupo BioBD, vamos enfatizar a avaliação de ações de sintonia fina considerando tanto índices como visões materializadas hipotéticas. Ações de sintonia fina como mudanças de configuração, particionamento, reescrita de consulta, entre outras, não foram levadas em consideração.

Para criação dos índices hipotéticos, foram utilizadas estratégias já conhecidas para dois SGBDs, sendo eles o *PostgreSQL* e o *SQLServer*. Já para as visões materializadas, foi desenvolvida uma estratégia original para o *PostgreSQL*. O cálculo de custo e benefício das estruturas é feito com uma mistura de contribuições originais e abordagens conhecidas.

Como resultado da pesquisa é apresentada uma nova arquitetura para o processo de tomada de decisão. Também são apresentados os algoritmos desenvolvidos para o processo de tomada de decisão. Resultados experimentais das contribuições são apresentados, utilizando uma carga de trabalho com consultas do TPC-H e ações de sintonia fina recomendadas pela ferramenta *OnDBTuning*.

1.2

Motivação

Diversas pesquisas buscam resolver o problema sugerir ações de sintonia fina, de diferentes tipos, para uma carga de trabalho. Essas ações normalmente são alterações de configurações do SGBD, ou criação de estruturas de acesso como índices e visões materializadas. Entretanto, poucas buscam ser um mecanismo de apoio a decisão do DBA no momento que ele precisa escolher, entre um conjunto de ações disponíveis, as mais benéficas para sua carga de trabalho quando olhamos para a relação custo-benefício.

A escolha das ações de sintonia fina precisa ser fundamentada, ou seja, levar em consideração o ganho esperado da ação e comparar com o seu custo estimado, tanto de criação quanto de manutenção. Uma ferramenta que apoie o DBA nessa decisão, de forma a sugerir as ações mais recomendadas para sua carga de trabalho, do ponto de vista do custo-benefício, precisa também oferecer contexto suficiente para que o DBA consiga tomar essas decisões. Definiremos como contexto informações como:

- impacto de uma ação em cada uma das consultas da carga de trabalho
- impacto de uma ação na carga de trabalho completa
- custo estimado de criação e manutenção de cada ação

1.3

Definição do Problema

Dado o contexto das pesquisas de sintonia fina em sistemas de bancos de dados relacionais e os problemas enfrentados na prática pelos DBAs, este trabalho busca responder à seguinte questão:

"Como apoiar o DBA no processo de sintonia fina, de forma que o mesmo consiga tomar decisões melhores, e fundamentadas, de quais ações de sintonia fina devem ser executadas?"

O processo de sintonia fina aqui considerado consiste em:

1. Coletar a carga de trabalho: coletar, em um período de tempo T , quais consultas foram executadas e algumas estatísticas das mesmas. Explicaremos carga de trabalho mais em detalhe no capítulo 2
2. Sugerir ações de sintonia fina
3. Avaliar as ações de sintonia fina possuem maior custo benefício
4. Aplicar as ações de sintonia fina no sistema de banco de dados

No capítulo 2 entraremos em mais detalhes sobre o que é a sintonia fina e suas etapas.

1.4

Estrutura da dissertação

A dissertação está organizada da seguinte forma:

- O capítulo 2 apresenta conceitos fundamentais para o entendimento do texto e trabalhos relacionados a esta pesquisa.
- O capítulo 3 apresenta a motivação da pesquisa com uma proposta de solução e seu requisitos.
- O capítulo 4 descreve as etapas relacionadas ao desenvolvimento da *TuningChef*.
- O capítulo 5 demonstra os resultados experimentais.
- O capítulo 6 apresenta a conclusão e descreve possíveis trabalhos futuros.
- O apêndice A lista as consultas TPC-H utilizadas nos resultados experimentais.
- O apêndice B lista as ações de sintonia fina utilizadas nos resultados experimentais.

2

Conceitos Básicos e Trabalhos Relacionados

Este capítulo apresenta conceitos básicos, porém fundamentais, para um melhor entendimento desta dissertação. Aqui também será apresentado o contexto da pesquisa e trabalhos relacionados.

2.1

Carga de Trabalho

Para a leitura dessa dissertação, vamos definir a carga de trabalho de um banco de dados como sendo o conjunto de consultas executadas em um período de tempo T [1]. O custo total de uma carga de trabalho pode ser formalizado da seguinte forma:

$$W_T = \sum_{i=1}^n C_i * R_i$$

onde

W : o custo total da carga de trabalho

T : período de coleta da carga de trabalho

n : número de consultas

C : custo de execução da consulta i

R : número de vezes que a consulta i foi executada

Apesar de não presente na fórmula acima, importante ressaltar que o custo C de uma consulta pode variar de acordo com o volume de dados em um banco. Uma mesma consulta pode ser muito rápida no presente mas, conforme novos dados são inseridos, ela se torna problemática. Isso significa que o W_T tem uma dependência implícita do volume de dados existentes no banco. Essa dependência não é levada em consideração nesse trabalho.

2.2

Sintonia Fina

Sintonia fina é o processo pelo qual um DBA busca reduzir o tempo de resposta das consultas em um banco de dados ou conseguir um aumento de vazão (transações por segundo, TPS) [2], podendo atingir ambos os objetivos simultaneamente. Este processo normalmente é feito através de uma seleção de

ações de *tuning*, propostas por um especialista de banco de dados, após uma análise crítica da carga de trabalho [1, 3].

As ações de sintonia fina envolvem, entre outros, ajustes de configurações e parâmetros do SGBD e mudanças em seu projeto físico, ou criação de novas estruturas de acesso como índices e visões materializadas, descritas em seções posteriores.

Toda ação de sintonia fina executada tem como objetivo ajudar o otimizador do SGBD a tomar melhores decisões, montando um plano de execução mais eficiente para a consulta. Isso significa que alterações do hardware por si só não são ações de sintonia fina, por mais que possam resultar em um melhor desempenho, pois o plano de execução permanece o mesmo. Outro ponto de atenção é que, como o processo de sintonia fina é feito de forma *online* sem desligar o banco, e interromper um sistema em produção para alterações de hardware muitas vezes não é viável, sendo essencial conseguir o melhor desempenho possível com o hardware disponível.

O *tuning* tem como objetivo influenciar nas decisões do otimizador de consultas e pode ser feito a qualquer momento, diferente da otimização que ocorre sempre que uma consulta é executada. Toda vez que uma consulta é executada, o otimizador tem pouco tempo para montar o plano de execução, e ele pode optar por não otimizar se entender que esse processo seria demorado (ou custoso) demais. Já a sintonia fina pode ser feita a qualquer momento e, inclusive, a decisão de quando realizá-la é um ponto de preocupação. Um exemplo é a tomada de decisão sobre quando criar um índice, uma vez que ele pode pegar um *lock* nos dados e, em horários de pico, isso pode derrubar uma operação pelas consultas sendo executadas não conseguirem ter acesso a tabela.

2.3

Índices

Um índice em um SGBD é uma estrutura de acesso, representado por diferentes estruturas de dados, capaz de representar dados de forma a otimizar determinados tipos de acesso às informações. Um índice é composto por uma chave, que pode referenciar uma ou mais colunas da tabela, e permite um acesso mais rápido aos dados se as condições da consulta fizerem uso dessas colunas [1, 4].

As estruturas dos índices permitem encontrar informações no banco de dados de forma mais eficiente, garantindo uma redução do consumo de *I/O* necessário para se chegar a informação requisitada. Um índice cria um mapa entre os valores de uma (ou mais) chave e suas tuplas correspondentes na

tabela onde foi criado [5].

Comumente os índices fazem uso de uma árvore B+, onde cada nó da árvore possui o valor da chave e uma referência para uma tupla na tabela. Índices também podem ser estruturados de outras maneiras como:

- tabela *hash*, onde cada entrada contém o *hash* da chave[29]
- árvore R, utilizada para indexação de dados em múltiplas dimensões[6]

Durante esta pesquisa foi feito o uso de índices hipotéticos, onde apenas os metadados dos índices existem no banco de dados, de forma que é possível simular o comportamento de uma consulta caso o índice fosse efetivamente criado[5, 9].

Além de diferentes estruturas de dados, os índices também podem ser classificados em diferentes categorias com diferentes casos de uso.

2.3.1

Índices Parciais

Índices parciais são muito comuns quando apenas uma parte dos dados de uma tabela precisa ser indexados[7]. Esse tipo de índice tem sido usado de forma recorrente como proposta de ação de sintonia fina[13, 14].

Uma situação para seu uso é o caso de dados que se comportam como séries temporais, onde o usuário normalmente realiza consultas analíticas que levam em consideração apenas dados mais recentes, então um índice que indexe apenas dados recentes é mais recomendado para essa situação. Em SGBDs compatíveis isso pode ser feito através da adição de uma cláusula *where* no comando de criação do índice.

A maior vantagem desse tipo de índice é que, por ser menor, sua chance de caber inteiramente na memória RAM é maior, o processo de limpeza feito pelos bancos também fica mais rápido e encontrar uma informação neles também.

2.3.2

Índices Compostos

Quando uma consulta realiza uma busca filtrando por duas ou mais colunas, temos duas estratégias comuns de indexação: criar um índice para cada coluna filtrada ou criar um índice único que envolva todas as colunas presentes no filtro, o qual chamamos de índice composto[8]. Se optarmos por criar N índices, sendo N o total de colunas presentes nos filtros, é necessário realizar uma operação de busca em cada um dos índices para depois fundir os resultados.

Já em um índice composto é feito a busca em uma única árvore B+, não sendo necessário fundir o resultado de diferentes buscas, tornando o processo muito mais rápido e otimizado quando comparado aos índices “simples”.

2.4

Visões

Uma visão é o resultado de uma consulta pré-compilada e aplicada sobre os dados já existentes no banco de dados. Muitas vezes é utilizada para facilitar o acesso a dados derivados de consultas mais complexas, evitando reescrever a mesma consulta várias vezes. Uma visão materializada é equivalente a uma visão, porém o seu resultado é persistido em disco.

O uso de visões materializadas é muito comum quando falamos de sintonia fina, uma vez que a mesma pode ser utilizada sem que os dados representados sejam recalculados[1, 10].

Um contraponto importante do uso de visões materializadas é o seu custo de criação e manutenção. Como os dados são armazenados em disco, é necessário ter espaço suficiente para sua criação e espaço de sobra para sua manutenção. Além do uso de disco, existe o custo atrelado a executar a consulta base responsável por manter as informações atualizadas, uma vez que alterações nas tabelas referenciadas podem tornar a visão materializada desatualizada[11].

De forma equivalente aos índices hipotéticos, visões materializadas hipotéticas são utilizadas ao longo dessa pesquisa, que nos permitem simular o custo de uma consulta caso a visão materializada fosse efetivamente criada. Diferente de índices hipotéticos, visões materializadas hipotéticas são muito pouco exploradas e, na prática, são muito úteis para avaliar o benefício de criar a estrutura quando levamos em consideração seus pontos negativos apresentados anteriormente. Um exemplo de seu uso é quando queremos avaliar se uma consulta que realiza a agregação de muitos dados históricos pode ser melhorada com uma visão materializada ou não, o que pode ser avaliado sem criar a visão efetivamente, utilizando seu formato hipotético.

2.5

Heurística de Benefícios

A heurística de benefícios[12] propõe a atribuição de benefícios às estruturas de acessos criadas (ex: visões materializadas, índices) de forma a ter uma métrica do quão benéfica, para uma carga de trabalho específica, aquela estrutura se mostrou ser quando comparada ao seu custo. Essa medida pode ser feita uma vez que é obtido o novo custo de uma carga de trabalho caso a

estrutura hipotética fosse criada e esse valor é comparado ao custo de criação e manutenção da estrutura.

Como exemplo da aplicabilidade dessa heurística, podemos considerar um índice que traz ganhos elevados para uma carga de trabalho mas seu custo não é desprezível, utilizamos da heurística para mensurar se o seu custo de criação e manutenção supera seus ganhos na carga de trabalho ou não. Em um caso oposto, se temos um índice que traz pequenos ganhos para a carga de trabalho, com custos de criação e manutenção elevados e maiores que o ganho, ele não vale a pena segundo a heurística de benefícios.

A heurística de benefícios foi utilizada em outras pesquisas de sintonia fina[3, 5, 15, 16, 17, 18]. Sua contribuição principal é a atribuição de benefícios às estruturas hipotéticas de forma a se chegar a uma conclusão se a estrutura real deve ser criada ou não.

2.6

Contexto e Trabalhos Relacionados

Sintonia fina de bancos de dados relacionais é um processo que vem tomando cada vez mais destaque devido ao crescimento do volume de dados coletados e armazenados globalmente e a necessidade de consultar esses dados de forma eficiente. Dessa forma, a constante melhora do processo de sintonia fina se mostra cada vez mais necessária. Mesmo com essa latente necessidade, o processo de sintonia fina ainda é pouco formal e considerado por alguns como *artístico*, por envolver profissionais experientes que, nesse processo, consideram seus conhecimentos adquiridos através da experiência, e não necessariamente se apoiam em práticas sistemáticas ou resultados científicos para a tomada decisão. Nesse cenário é comum que, para problemas idênticos, DBAs distintos apresentem soluções diferentes.

Existem várias dificuldades que o DBA enfrenta nesse processo, como a necessidade de ter um conhecimento profundo da natureza dos dados armazenados, conhecimento o qual muitas vezes fica concentrado em times que o DBA não possui contato suficiente para ter.

Pesquisas na área, inclusive do grupo BioBD da PUC-Rio, têm trabalhado em soluções para facilitar e automatizar o processo de sintonia fina e o trabalho do DBA. Dentre estas podemos destacar os trabalhos feitos por Marcos Salles[3] e Eduardo Morelli[15], que recorreram ao uso de agentes de *software* para a automatização de atividades relacionadas ao processo de sintonia fina de índices. O trabalho do José Maria[25], por exemplo, faz o uso de estratégias automáticas para a manutenção do projeto físico de um banco de dados de forma não intrusiva, utilizando índices, visões materializadas, par-

ticionamento e duplicação de estruturas físicas. Como diferença do trabalho proposto, as pesquisas citadas não permitem a integração do conhecimento do DBA na solução, nem que o mesmo tome decisões sobre o que deve ser feito no final do processo.

Outra pesquisa que merece destaque é a de Rafael de Oliveira, que propôs em seu trabalho estratégias de combinação de ações de sintonia fina através do uso da técnica de *branch and bound*[26]. Este trabalho buscou resolver o problema de responder quais ações de sintonia fina podem ser combinadas, evitando que ações incompatíveis entre si sejam executadas em conjunto. Um exemplo é uma visão materializada que, se criada, faz com que um índice criado deixe de ser utilizado.

No grupo de pesquisa do BioBD foi desenvolvido um *framework* chamado *Outer-Tuning*[27] com o propósito de facilitar o trabalho de um DBA. Tal *framework* propõe ações de sintonia fina como a criação de índices e visões materializadas, e faz uso da heurística de benefícios para determinar quais ações propostas devem ser executadas. O *framework* pode ser executado de forma automática, em que estruturas de acesso são sugeridas e criadas, e de forma semiautomática, oferecendo certo nível de controle sobre as escolhas ao DBA utilizando a ferramenta através de uma interface.

Outra ferramenta desenvolvida pelo grupo de pesquisa do BioBD é a *OnDBTuning*[28] que faz uso da ontologia com regras SPIN para inferir, dado uma carga de trabalho, um conjunto de ações de sintonia fina aplicáveis como a criação de índices e visões materializadas.

Pesquisas mais recentes na área de sintonia fina propõe o uso de técnicas de aprendizado de máquina (*Machine Learning* - ML) de diferentes formas. Dentre elas podemos citar:

- OtterTune: ferramenta de *tuning* automático para PostgreSQL e MySQL que, através de métodos supervisionados e não supervisionados de aprendizado de máquina, constantemente monitora o SGBD e altera suas configurações[19]. Diferente da solução proposta neste trabalho, a OtterTune trabalha apenas com alteração de configurações, e não permite a tomada de decisão por parte do DBA.
- Redes neurais como estruturas de dados para índices. Em um estudo realizado por pesquisadores da Google, resultados preliminares no uso de redes neurais como substitutos para árvores B+, obteve-se um ganho de 44% de performance utilizando 2/3 de memória[20].
- CDBTune+: ferramenta que, através de técnicas de *Deep Reinforcement Learning*(RL) e práticas de tentativa e erro, aprende como as diferentes

configurações de um SGBD impactam a carga de trabalho e recomenda valores otimizados para cada uma das configurações disponíveis[21]. Diferente da solução proposta neste trabalho, a CDBTune+ trabalha apenas com alteração de configurações, e não permite a tomada de decisão por parte do DBA.

- Desenvolvimento de uma inteligência artificial capaz de classificar planos de execução das consultas, com o objetivo de fornecer melhores recomendações de índices, evitando recomendações que podem vir a piorar a execução das consultas[22].

O trabalho aqui descrito busca propor e desenvolver uma ferramenta de recomendação e apoio a decisão em uma das etapas do processo de sintonia fina, a etapa de decisão do DBA onde dado uma carga de trabalho e um conjunto de ações de sintonia fina, como escolher as opções de melhor custo-benefício. Diferente dos trabalhos descritos anteriormente, a proposta não busca sugerir ações de sintonia fina a partir de uma carga de trabalho, mas sim analisar um conjunto de ações de sintonia fina e recomendar, dentre elas, as com melhor relação custo-benefício para a carga de trabalho.

Outro ponto importante de diferença entre o trabalho proposto e os relacionados é a possibilidade do DBA de aplicar o seu conhecimento na tomada de decisão. Apesar do resultado final conter uma sugestão de quais ações executar, o DBA tem a possibilidade de descartar as recomendações e tomar as próprias decisões, visualizando o impacto esperado na carga de trabalho das suas escolhas. A participação do DBA é parcialmente presente na *Outer-Tuning*, onde o administrador podia escolher quais heurísticas seriam executadas, mas não era permitido tomar decisões no nível de granularidade das ações em si.

É importante que esse processo de decisão apresente contexto suficiente de forma que o DBA entenda a motivação por trás de cada decisão tomada, contexto o qual é pouco presente, ou não existe, nas pesquisas mencionadas. Esse contexto deve apresentar o impacto de uma ação de sintonia fina em cada consulta da carga de trabalho, e também na carga total. Sem esse contexto, o processo de decisão torna-se vago para um DBA, abrindo a porta para questionamentos como: "por que o índice A deve ser criado e não o B?" ou "por que criar a visão materializada A e não os índices B e C?", entre outros. Esses questionamentos, se presentes, podem levar ao entendimento de que a ferramenta está apresentando soluções ruins, do ponto de vista do DBA, por apresentar soluções que não estão de acordo com o que o mesmo esperava baseado em suas experiências no assunto.

2.7

Resumo do Capítulo

Neste capítulo foram apresentados os conceitos e fundamentos necessários para contextualizar a pesquisa sendo apresentada.

No capítulo 3 a seguir será apresentada em detalhes a pesquisa, seus requisitos funcionais e suas etapas de desenvolvimento.

3

Proposta de solução

O presente capítulo apresenta a motivação por trás da solução, através de um exemplo de ciclo de vida do processo de sintonia fina, fazendo ligações entre as etapas envolvidas com trabalhos existentes. Apresenta-se também espaços existentes para contribuição na etapa de tomada de decisão, a qual deve analisar os custos e benefícios para cada uma das possíveis ações de sintonia fina. Além disso apresentamos requisitos funcionais para a solução e uma proposta de arquitetura para seu desenvolvimento.

3.1

Motivação

Definir as melhores estratégias de sintonia fina é um trabalho extremamente complexo. Muitos fatores precisam ser levados em consideração: o DBA precisa fazer uma análise minuciosa da carga de trabalho, ter conhecimento do domínio da aplicação, ser experiente com os diferentes SGBDs existentes e conciliar todo esse conhecimento em um conjunto de possíveis ações de sintonia fina e escolher, dentre elas, as melhores para seu caso.

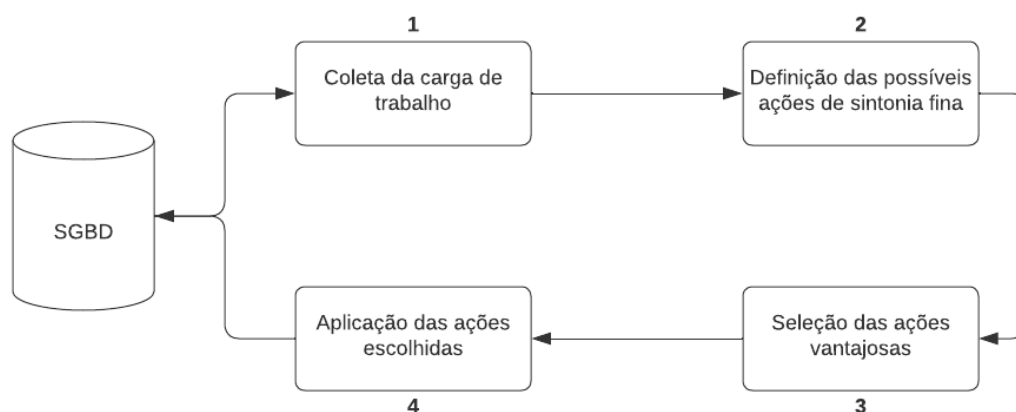


Figura 3.1: Exemplo de um processo de sintonia fina

Na Figura 3.1 temos um exemplo de um processo de sintonia fina onde, em todas as etapas, temos o DBA como componente fundamental. No início do

processo é feita uma coleta da carga de trabalho (1), a qual é posteriormente analisada com fins de definir quais ações de sintonia fina podem vir a contribuir para a redução do custo total da carga coletada (2). Com o conjunto de ações em mãos, é necessário fazer uma análise do impacto que cada ação pode trazer, visto que elas possuem um custo atrelado, e comparar com seu benefício caso aplicada (3). Após essa análise as ações são aplicadas no SGBD (4).

Diversas soluções existentes buscam resolver o problema de mapear as ações de sintonia fina disponíveis (etapa 2), de forma a sugerirem alteração nas configurações do SGBD[19, 20, 21] ou criação de novas estruturas de acesso[27, 28]. No entanto, quando olhamos para soluções com o objetivo de apoiar o DBA no momento da tomada de decisão (etapa 3), são poucas e, as existentes, se restringem a apoiar na escolha apenas entre as ações que ela mesma sugeriu. Isso ocorre pelo fato do processo de decisão ser pouco sistemático e baseado no conhecimento do DBA, conforme descrito na seção 2.6, e necessitar de um conhecimento profundo do domínio da carga de trabalho para poder estabelecer uma relação de custo-benefício de cada uma das possíveis ações.

Para tomar uma decisão bem fundamentada, isto é, uma decisão que leve em consideração o impacto local (em cada consulta), global (em toda carga de trabalho), e o custo de criação e manutenção das ações de sintonia fina, é importante que seja apresentado ao DBA informações suficientes no momento da tomada de decisão. Essa visão é fundamental, uma vez que algumas ações de sintonia fina que podem ser custosas em uma primeira análise, mas quando avaliadas na carga de trabalho completa, podem valer a pena. Um exemplo disso é a criação de um índice composto para uma consulta que, após validar seu benefício, descobre-se que na verdade aquele índice pode ser reaproveitado por outras consultas que não apenas a original, fazendo-o valer a pena.

No momento de decisão também é importante que o administrador consiga decidir entre diversas soluções de sintonia fina, não se restringindo apenas as soluções propostas por uma ferramenta específica. Para isso, é necessário que no momento da decisão, o DBA consiga comparar o impacto de ações de sintonia fina provenientes de várias fontes, sejam elas ferramentas terceiras ou ações sugeridas por ele mesmo.

O conhecimento do DBA no processo de tomada de decisão é de extrema relevância, o que significa que a decisão final sobre o que deve, ou não, ser feito, pode ser dele, e não obrigatoriamente automatizado. O mesmo pode ter conhecimento de mudanças futuras na carga de trabalho que beneficiariam certas ações de sintonia fina que não foram recomendadas inicialmente, optando seguir por um plano diferente do recomendado.

Além disso, muitas dessas ferramentas propostas[17, 18, 19, 21, 28] são feitas com um escopo fechado, ou seja, elas sugerem ações de *tuning* e apresentam o benefício de cada de suas próprias sugestões, sem permitir que o DBA adicione suas próprias, que o mesmo considera como boas opções para serem avaliadas em conjunto com as demais.

Dado esse cenário, o presente trabalho propõe uma ferramenta para o apoio à decisão do DBA, com o objetivo de recomendar ao administrador quais ações de sintonia fina devem ser executadas, com contexto suficiente para que o operador entenda a motivação de cada decisão. Nesse método, é levada em consideração a participação e liberdade do DBA de tomar decisões diferentes das propostas e avaliar o impacto de suas escolhas na carga de trabalho completa.

3.2

Requisitos funcionais

Durante a definição do escopo da solução, foram levantados os requisitos desejáveis, sendo eles:

- Capturar a carga de trabalho: para fazer a análise do custo benefício das ações de sintonia fina, é necessário ter a carga de trabalho como fonte de dados para a análise.
- Ser extensível para diferentes SGBDs: soluções existentes muitas vezes se restringem a SGBDs específicos, normalmente *Oracle* e *SQLServer*. Funcionar para diferentes sistemas de bancos de dados torna a solução mais atrativa para adoção pública, principalmente em ambientes que fazem uso de diferentes sistemas de bancos de dados relacionais no dia a dia.
- Ser extensível para diferentes estratégias de *tuning*: de forma a cobrir cenários de sintonia fina diferentes, deve ser fácil criar estratégias de avaliação para diferentes tipos de ações de sintonia fina, desde ações generalizadas como índices e visões materializadas, como também para ações restritas a certos SGBDs, como por exemplo o *Column Tetrís* no caso do *PostgreSQL*[35]. Extensibilidade aqui é a capacidade de funcionar com diferentes SGBDs e ser simples de adicionar integrações com novos bancos de dados no futuro.
- Utilizar estratégias de *Branch-and-Bound* (BnB) para avaliar as ações de sintonia fina: quando está sendo avaliado um grande conjunto de ações de sintonia fina, o espaço de busca cresce exponencialmente e o uso do BnB contribui para diminuir o espaço de busca. O BnB é um paradigma

para design de algoritmos para otimização de problemas de combinação. Neste paradigma existe o componente de *bound*, utilizado para verificar se um galho da árvore está de acordo com as expectativas e deve continuar sendo explorado ou não. Para o contexto do trabalho, o *bound* é o ganho da ação de sintonia fina e, se ele for zero, o galho deixa de ser explorado.

- Aceitar ações de sintonia fina de diferentes fontes: como existem muitas ferramentas capazes de propor ações de sintonia fina diferentes, algumas dedicadas a configuração de parâmetros e outras dedicadas a estruturas de dados, unificar todas as opções em um ponto central permite oferecer ao DBA uma avaliação mais detalhada sobre o que seria benéfico, ou não, de ser executado de uma forma mais ampla. A independência das fontes de ações de *tuning* também permite ao DBA comparar soluções propostas por ele mesmo junto a soluções propostas por outras ferramentas. Extensibilidade aqui é a capacidade de analisar diferentes tipos de ações de sintonia fina, e ao mesmo tempo se simples adicionar estratégias para avaliar diferentes tipos de ações de sintonia fina no futuro.
- Exibir contexto para que as decisões sejam claras: as decisões por si só são benéficas ao administrador, mas quando somado ao contexto oferecido, o DBA consegue entender a motivação por trás de cada decisão e fazer uma avaliação se concorda ou não. O contexto é um conjunto de informações que devem ser disponibilizados para cada ação de sintonia fina avaliada, como o impacto estimado em cada consulta e o custo de criação de manutenção esperados. Em casos extremos, o DBA pode discordar totalmente das escolhas por ter informações futuras de mudança na carga de trabalho, e optar por tomar as próprias decisões baseado em seu conhecimento. Esse contexto deve oferecer informações suficientes para que um DBA consiga entender, por exemplo, porquê o índice A foi escolhido, mas não o B.

Baseado nos requisitos levantados, propõe-se desenvolver uma ferramenta denominada *TuningChef*, de forma que a mesma seja capaz de atender os requisitos acima listados e explorar etapas do processo de sintonia fina ainda pouco explorados.

3.3

Projeto de arquitetura da *TuningChef*

Para o desenvolvimento da ferramenta foi importante considerar, primeiramente, quais módulos precisariam ser desenvolvidos e como eles deveriam

comunicar entre si e com o meio externo. Para isso usou-se como base a arquitetura do arcabouço *Outer-Tuning*[18], apresentado na imagem abaixo.

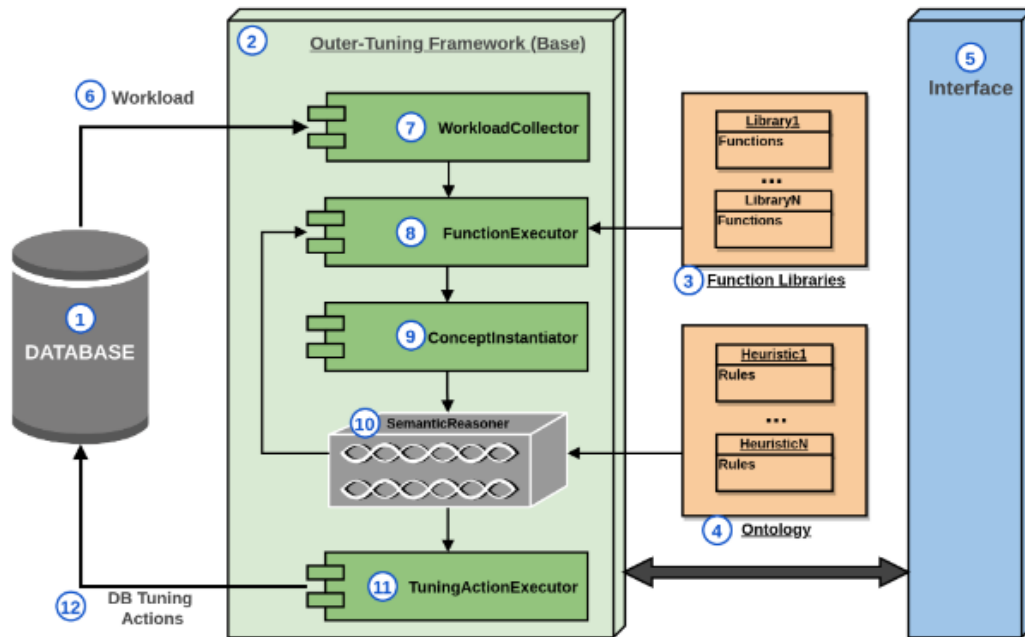


Figura 3.2: *Arquitetura Outer-Tuning*

A figura 3.2 apresenta a arquitetura da *OuterTuning*, onde temos:

1. O banco de dados sendo avaliado.
2. Biblioteca base de funções compartilhadas entre os componentes.
3. Código compilado de bibliotecas responsáveis por extrair os conceitos da carga de trabalho.
4. Contém os conceitos instanciados pela carga de trabalho e as heurísticas utilizadas.
5. Responsável pela interação com o DBA.
6. A carga de trabalho coletada, formada pelo conjunto de consultas executadas, com seus planos de execução e frequência de execução.
7. Responsável por coletar a carga de trabalho do banco de dados relacional.
8. Responsável por extrair informações da carga de trabalho e gerar as instâncias dos conceitos da ontologia.

9. Instancia as pré-condições geradas pela execução da etapa anterior na ontologia.
10. Componente responsável pela seleção e execução das regras definidas na ontologia.
11. Captura as ações de tuning sugeridas e as executa no SGBD de acordo com as escolhas do DBA.
12. As ações de sintonia fina executadas no SGBD.

Quando a arquitetura apresentada na Figura 3.2 foi comparada com os requisitos apresentados anteriormente, deparou-se com alguns problemas e mudanças foram necessárias. A *Outer-Tuning* avaliava o impacto das soluções que ela propunha, mas não permitia comparar com ações de sintonia fina do próprio DBA ou de outras ferramentas, o que já deixa de atender um dos requisitos enunciados.

Outro ponto é que a *Outer-Tuning* faz uso da ontologia para sugerir ações de sintonia fina, porém esse comportamento estava fixado para apenas uma carga de trabalho como prova de conceito, e alterar a ontologia se mostrou um trabalho muito custoso. Em contrapartida, pode-se continuar utilizando ontologia através da *OnDBTuning*[28] como uma das fontes de ações de sintonia fina da *TuningChef*. Na área de computação define-se ontologia como sendo um conjunto de primitivas representacionais que modelam um domínio de conhecimento ou discurso. As primitivas são tipicamente classes (ou conjuntos de objetos), atributos (ou propriedades) e relacionamentos (ou relações entre membros de classes)[23].

Dessa forma, a seguinte arquitetura foi proposta para o desenvolvimento da *TuningChef*:

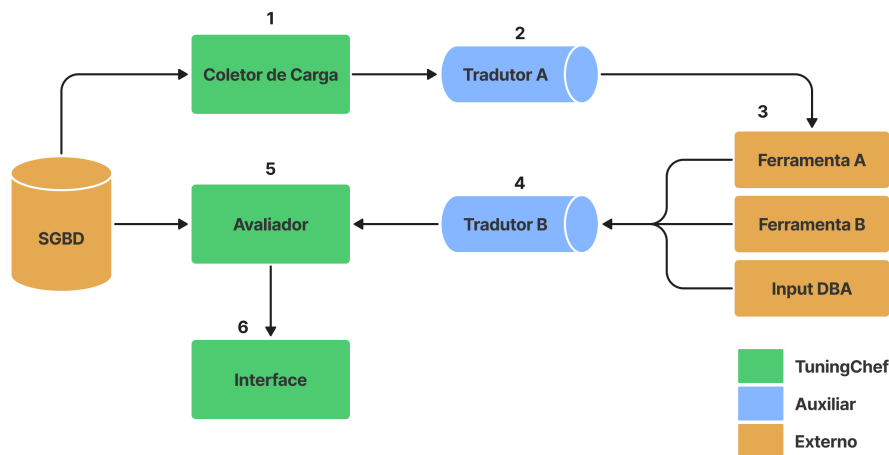


Figura 3.3: Arquitetura TuningChef

A arquitetura definida para a *TuningChef* (Figura 3.3) tem início com a coleta da carga de trabalho do banco de dados (1) que, uma vez feita, pode ser fornecida a um ou mais tradutores (2) responsáveis por transformar a carga de trabalho em um formato textual que outras ferramentas podem ler (3). As ferramentas que fazem uso da carga de trabalho por sua vez são responsáveis por processar essas informações e sugerir ações de sintonia fina, ações que são então traduzidas (4) para um formato que a camada de avaliação espera. O avaliador realiza a leitura desse conjunto de ações (5) e é responsável por agregar, filtrar e ordenar esse conjunto, utilizando estruturas hipotéticas e os custos providos pelo SGBD. Com o processo finalizado, o resultado final fica disponível para o DBA através de uma interface (6), onde o mesmo pode avaliar o plano de ação recomendado ou montar o seu próprio, para então executar o conjunto de ações no banco de dados. O plano de ação é o conjunto de ações de sintonia fina a serem executadas, e sua ordem. A habilidade de montar o seu próprio plano de ação é fundamental para integrar o conhecimento do DBA na ferramenta, enriquecendo o fluxo, e para montá-lo o DBA precisa, no final, apenas selecionar quais ações deseja executar. O mesmo pode ter informações sobre mudanças futuras na carga de trabalho, e com elas decidir por antecipar algumas ações de *tuning* que acredita ser benéficas para essas mudanças.

Ao comparar as figuras 3.3 e 3.2, é importante destacar suas diferenças e os benefícios que a arquitetura proposta (Figura 3.3) traz. Diferente da *Outer-Tuning*, a *TuningChef* não se restringe a apenas uma fonte de ações

de sintonia fina, uma vez que ela não é responsável por criar essas sugestões, mas apenas avaliá-las. Isso dá a ferramenta a flexibilidade de comparar ações de sintonia fina de diferentes fontes, o que não era possível anteriormente. Essa comparação é importante pois diferentes fontes podem usar estratégias distintas de sugestão, resultando em diferentes ações de sintonia fina que podem ser complementares entre si, ou excludentes.

Outro fator importante é como ambas as ferramentas integram o DBA em seu uso. A *Outer-Tuning* permitia o DBA escolher quais heurísticas de sugestão utilizar. Já a *TuningChef* oferece ao DBA uma visão global do impacto de cada ação de sintonia fina, bem como um visão de impacto acumulado esperado do conjunto de ações de sintonia fina escolhidas, permitindo que o DBA escolha, dentre todas as ações avaliadas, quais devem ser executadas e em qual ordem. Esse controle permite uma granularidade maior no nível de controle das ações que serão aplicadas no sistema de banco de dados.

3.4

Conclusão

Este capítulo apresentou a motivação por trás da necessidade de uma solução para apoiar a decisão do DBA no processo de sintonia fina e apresentou uma proposta de arquitetura para o desenvolvimento de uma ferramenta, a *TuningChef*.

No próximo capítulo será apresentado o processo de desenvolvimento da ferramenta, as soluções utilizadas e os algoritmos desenvolvidos.

4

Desenvolvimento da *TuningChef*

O presente capítulo apresenta o processo de desenvolvimento por trás da *TuningChef*, explicitando algoritmos e tecnologias utilizadas.

4.1

Tecnologias utilizadas

Para o desenvolvimento dos módulos necessários foi escolhido o uso da linguagem de programação *Python*, versão 3.10. De forma a garantir o controle de qualidade do código foram utilizadas ferramentas adicionais como *MyPy* e *Black*.

O *MyPy*¹ é uma ferramenta desenvolvida originalmente pela empresa *Dropbox* que permite realizar uma checagem estática dos tipos utilizados em um projeto escrito em *python*. Essa análise torna mais fácil encontrar problemas derivados do uso de tipos diferentes dos esperados em uma linguagem dinâmica, além de facilitar a manutenção de um projeto e garantir que todas as variáveis tem um tipo explícito. A prática de tipagem explícita em *python* vem aumentando nos últimos anos e seus benefícios tem sido notados na comunidade[30, 31, 32].

O *Black*² é uma ferramenta responsável por fornecer uma opinião forte e fundamentada sobre como um código em *python* deve ser formatado. Ao adotá-la, garantimos que o mesmo padrão é utilizado em todo o projeto, sem existir divergências no formato do código.

4.2

Coletor de Carga

O primeiro módulo da *TuningChef* presente na arquitetura proposta (Figura 3.1) é o coletor de carga. Este módulo é responsável por coletar a carga de trabalho de um banco de dados de forma a oferecer ao DBA, e a outras ferramentas interessadas, as seguintes informações:

- Consultas executadas durante a coleta
- Número total de execuções de cada consulta

¹<https://github.com/python/mypy>

²<https://github.com/psf/black>

- Plano de execução das consultas

Além das informações listadas acima, o coletor também provém o esquema do banco, sendo ele a lista de tabelas existentes, com informações de suas colunas e índices.

Este módulo tem, como requisito funcional, que ser extensível para diferentes SGBDs. Para fins de demonstração foram escolhidos os SGBDs *PostgreSQL* e *MySQL*.

4.2.1

Arquivo de entrada

Para a execução do coletor de carga é necessário fornecer um arquivo de configuração inicial. Foi escolhido o formato de arquivo *toml* devido a sua popularidade na comunidade *Python*. Neste arquivo o usuário escolhe qual o SGBD será utilizado na coleta de carga e coloca suas configurações de acesso, sendo elas:

- usuário
- senha
- endereço IP e porta
- nome do banco (*database*)

No mesmo arquivo o usuário pode configurar por quanto tempo a coleta será feita e o formato de saída dos dados.

4.2.2

Formatos de saída

Duas opções de saída dos dados foram desenvolvidas, sendo elas as opções *sqlite* e *json*.

Caso o usuário escolha pela opção *sqlite*, terá como resultado da coleta um arquivo *collection.db*, o qual pode ser acessado através do uso do *sqlite* para fazer análises na carga de trabalho.

Já na opção *json*, o resultado da coleta de carga fica disponível em um arquivo *collection.json*, o qual pode ser usado como entrada para outras ferramentas. Como exemplo, esse arquivo pode ser traduzido para o formato esperado pela *OnDBTuning* para obter sugestões de ações de sintonia fina.

4.2.3

Algoritmo de coleta

Algorithm 1 Algoritmo para coleta da carga de trabalho (CCT)

Entrada: *dsn, saida, ciclos, intervalo*

```

1: início
2:   coletor  $\leftarrow$  novo_coletor(dsn)
3:   repositorio  $\leftarrow$  novo_repositorio(saida)
4:   contador  $\leftarrow$  0
5:
6:   while contador  $\leq$  ciclos do
7:     fotografia  $\leftarrow$  coletor.coletar()
8:     repositorio.salvar(fotografia)
9:     dormir(intervalo)
10:    contador  $+$  = 1
11:  end while
12:
13:  coletor.finalizar()
14:  repositorio.finalizar()
15: fim

```

No algoritmo desenvolvido (Algoritmo 1) para a coleta da carga de trabalho recebemos como entrada as configurações necessárias para sua execução, sendo:

1. *dsn*: as configurações de acesso ao SGBD
2. *saida*: a escolha do formado de saída da coleta
3. *ciclos*: o número de ciclos de coleta
4. *intervalo*: o intervalo em segundos entre cada ciclo da coleta

Com as variáveis de entrada, instanciamos o coletor para o SGBD escolhido (linha 2) e o repositório para o formato de saída escolhido (linha 3). Em seguida, iniciamos um contador com o valor 0 (linha 4) responsável por manter atualizado o número de ciclos de coleta executados.

A seguir iniciamos a etapa principal da coleta onde, para cada ciclo, é pedido ao coletor uma *fotografia* do estado atual do banco, a qual é repassada para o repositório salvar e então esperamos pelo intervalo desejado e atualizamos o contador (linhas 6 - 11).

Por último, quando o número de ciclos desejado já foi coletado, informamos ao coletor e ao repositório que finalizem seus trabalhos (linhas 13 - 14).

4.2.4

PostgreSQL

A implementação do coletor para o *PostgreSQL* executa uma consulta inicial para extrair informações a partir do esquema físico do banco de dados no começo de sua execução. O SQL executado é apresentado na Consulta 4.1.

```
1 with tables as (  
2     select schemaname,  
3         tablename,  
4         tableowner  
5     from pg_catalog.pg_tables  
6     where schemaname != 'pg_catalog'  
7         and schemaname != 'information_schema'  
8 )  
9 select  
10     table_schema || '.' || table_name as table_name,  
11     json_agg(  
12         json_build_object(  
13             'name', column_name,  
14             'type', udt_name,  
15             'nullable', is_nullable  
16         )  
17     ) as columns  
18 from tables tbs  
19     join information_schema.columns cls  
20         on cls.table_schema = tbs.schemaname  
21         and cls.table_name = tbs.tablename  
22 where cls.table_catalog = $1  
23 group by 1;
```

Consulta 4.1: Esquema físico PostgreSQL

A consulta 4.1 busca no catálogo do *PostgreSQL*, para cada tabela existente, o nome do esquema em que está inserida e informações de suas colunas, sendo elas o nome, o tipo de dados e se pode conter valor nulo ou não.

Para a coleta da carga no *PostgreSQL* são feitas duas consultas, sendo a primeira para pegar as consultas em seu formato genérico e a segunda para

pegar as consultas com seus parâmetros.

As consultas com seus parâmetros podem ser resgatadas da visão *pg_stat_activity* disponibilizada no SGBD. Esta visão contém as consultas em execução no momento, ou seja, são temporárias e deixam de aparecer na visão uma vez que a consulta termina sua execução. De forma a capturar o máximo de consultas possíveis, a captura é feita em uma corrotina que é executada de forma independente do *loop* principal do Algoritmo 1, e de forma ininterrupta. Seu SQL é apresentado na Consulta 4.2.

```
1 select query
2   from pg_stat_activity
3  where user is not null
4     and query not like '%insufficient%'
5     and query <> '';
```

Consulta 4.2: Consultas Parametrizadas PostgreSQL

Para as consultas em seu formato genérico (sem os parâmetros) é utilizado um módulo disponibilizado pelo próprio SGBD chamado *pg_stat_statements*. Este módulo cria uma visão chamada também de *pg_stat_statements* que contém o restante das informações necessárias para a coleta das estatísticas das consultas. Como os dados apresentados nesta visão não são temporários como no caso anterior, a coleta de informações é feita de forma sincronizada com o loop principal (Algoritmo 1). Seu SQL é apresentado na Consulta 4.3.

```
1 select queryid,
2       query,
3       calls,
4       total_exec_time,
5       mean_exec_time
6 from pg_stat_statements
7 where dbid = (
8   select oid
9   from pg_database
10  where datname = $1
11 );
```

Consulta 4.3: Consultas Genéricas PostgreSQL

Ao final da coleta de carga, o coletor encerra a conexão com o banco de dados.

4.2.5 MySQL

A implementação do coletor para o *MySQL* executa uma consulta inicial para coletar informações do esquema físico do banco de dados no começo de sua execução. O SQL executado é apresentado na Consulta 4.4.

```
1 with tables as (  
2     select table_schema,  
3           table_name  
4     from information_schema.tables  
5     where table_type = 'BASE_TABLE'  
6     and table_schema not in (  
7         'mysql', 'performance_schema', 'sys'  
8     )  
9 )  
10 select concat_ws(  
11     '.', tbs.table_schema, tbs.table_name  
12 ) as table_name,  
13     json_arrayagg(  
14         json_object(  
15             'name', cls.column_name,  
16             'type', cls.column_type,  
17             'nullable', cls.is_nullable  
18         )  
19     ) as columns  
20 from tables as tbs  
21 join information_schema.columns cls  
22     on cls.table_schema = tbs.table_schema  
23     and cls.table_name = tbs.table_name  
24 group by 1;
```

Consulta 4.4: Esquema Físico MySQL

A consulta 4.4 busca no catálogo do *MySQL*, para cada tabela existente, o nome do esquema em que está inserida e informações de suas colunas, sendo elas o nome, o tipo de dados e se pode conter valor nulo ou não.

Para a coleta da carga no *MySQL* apenas uma consulta é necessária. O SGBD possui uma visão chamada *events_statements_history_long* que possui todas as informações necessárias para coletar as estatísticas das consultas executadas e as consultas em ambos os formatos, com seus parâmetros ou

no formato genérico. O SQL executado para coletar a carga é apresentado na Consulta 4.5.

```

1  select min(event_id) as id,
2      count(*)      as calls,
3      digest_text   as query,
4      sum(truncate(
5          timer_wait/1000000000000,6
6      ))            as total_exec_time,
7      avg(truncate(
8          timer_wait/1000000000000,6
9      ))            as avg_exec_time
10 from performance_schema.events_statements_history_long
11 where digest_text regexp %s
12 and digest_text not like %s
13 group by digest_text;
```

Consulta 4.5: Consultas MySQL

Ao final da coleta de carga, o coletor encerra a conexão com o banco de dados.

4.3 Avaliador

O segundo módulo da *TuningChef* presente na arquitetura proposta (Figura 3.1) é o avaliador e sua interface. Este módulo é responsável por, dado um conjunto de ações de sintonia fina e uma carga de trabalho, montar um plano de execução.

Para montar o plano de execução é necessário avaliar o custo-benefício de cada ação proposta e combinar as ações de forma a reduzir o custo total da carga de trabalho de forma que o custo-benefício seja máximo, ou seja, ter a maior redução de custo total com o menor custo de criação e manutenção das ações possível.

De forma a montar o plano de execução foi estudada a estratégia de *Branch-and-bound* (BnB) apresentada por Rafael em sua tese de doutorado[26], que por sua vez foi implementada com algumas modificações.

O *BnB* é um paradigma de design de algoritmo que busca enumerar, de forma inteligente, um subconjunto de ações ótimas para solução de um problema. O termo *branch* é utilizado para dar a ideia de que o conjunto total de ações é particionado em "galhos", em uma estrutura de árvore. Já o termo *bound* impõe restrições do problema que limitam o espaço de busca.

Para o caso de ações de sintonia fina, o BnB se mostra valioso por possibilitar o descarte de ações que não trazem benefícios para a carga de trabalho, reduzindo o espaço de busca e otimizando o algoritmo.

Este módulo tem, como requisito funcional, que ser extensível a diferentes SGBDs e ter a capacidade de avaliar diferentes soluções de sintonia fina. O seu resultado final também precisa conter contexto suficiente para que um DBA, ao avaliar o resultado, conseguir entender a motivação por trás de cada decisão tomada pelo módulo.

Para o desenvolvimento do avaliador foi necessário montar um arquivo de entrada e configuração de uso amigável, que permitisse ao usuário escolher o seu SGBD, junto de suas configurações de acesso, e fornecer as ações de sintonia fina que deseja avaliar para a carga de trabalho. Dessa forma, foi escolhido utilizar dois arquivos distintos, sendo um para configuração no formato *toml* e um arquivo de entrada para as ações no formato *json*.

Para montar o plano de ação das ações de sintonia fina, foi desenvolvido o seguinte algoritmo recursivo:

Algorithm 2 Algoritmo para seleção de ações de sintonia fina (SASF)

Entrada: N, S

Saída: N

```

1: início
2:    $\delta_{maximo} \leftarrow -\infty$ 
3:    $recomendado \leftarrow \emptyset$ 
4:   procedure SASF( $N, S$ )
5:     while  $i, a = S$  do
6:        $q\Delta, ganho, custo \leftarrow a.aplicar()$ 
7:       if  $ganho \leq 0$  then
8:         continue
9:       end if
10:
11:        $acoes = copia(S.pop(i))$ 
12:        $novo\_no = SASF(cria\_no(a, q\Delta, ganho, custo, N), acoes)$ 
13:
14:       if  $novo\_no.\delta > \delta_{maximo}$  then
15:          $\delta_{maximo} \leftarrow novo\_no.\delta$ 
16:          $recomendado \leftarrow novo\_no$ 
17:       end if
18:
19:        $N.adiciona\_filho(novo\_no)$ 
20:        $a.desfazer()$ 
21:     end while
22:     return  $N$ 
23:   end procedure
24: fim

```

No algoritmo recursivo desenvolvido (Algoritmo 2) recebemos como entrada duas variáveis, sendo N o nó raiz de uma árvore e S o conjunto de ações de sintonia fina disponíveis. Temos também duas variáveis globais, *delta_recomendado* e *recomendado*, que são usadas de forma a auxiliar a encontrar o nó com melhor custo-benefício

Começamos o procedimento iterando sobre as ações existentes em S (linha 5), atribuindo a i o índice da ação na lista e a a a ação em si. Para cada ação a a mesma é então aplicada (linha 6), retornando a diferença de custo de cada consulta afetada da carga de trabalho, o ganho total e o custo de aplicar a ação. Nesse caso aplicar uma ação é equivalente a criar uma estrutura hipotética no SGBD e avaliar seu impacto na carga de trabalho. Se seu ganho total for menor ou igual a 0, partimos para a próxima ação da lista (linhas 7-10).

Com a ação aplicada, ela é removida da lista original recebida (linha 11) e chamamos o procedimento de forma recursiva, passando um novo nó como sendo a raiz da árvore, o qual contém as estatísticas resultantes de termos aplicado a ação anteriormente, e a lista de ações ainda disponíveis (linha 12).

Quando a chamada recursiva retorna, verificamos se o *novo_no* possui um delta maior do que o *delta_maximo* e, caso tenha, atualizamos as variáveis e esse se torna nosso novo nó recomendado (linhas 14-17). O delta de um nó é a diferença entre seu ganho e custo, somado ao delta dos nós até a raiz da árvore. Em seguida adicionamos o novo nó como filho do nó pai (linha 19) e desfazemos a ação previamente aplicada (linha 20).

Por fim, quando não existem mais ações para serem aplicadas e nós para serem criados, retornamos o nó raiz da árvore, N , que representa a árvore de execução das ações (linha 22).

A árvore resultante é composta de nós onde cada um é uma representação de um estado da carga de trabalho caso a ação do mesmo, e dos nós até a raiz da árvore, fossem aplicados. Essas informações permitem oferecer ao usuário da ferramenta contexto suficiente para entender a motivação por trás da recomendação final. Além disso, se o usuário optar por montar o seu próprio plano de ação, ele consegue ter uma noção do impacto esperado de seu plano na carga de trabalho avaliada.

Para o desenvolvimento do avaliador foram escolhidos os bancos *PostgreSQL* e *SQLServer* como SGBDs utilizados, de forma a demonstrar a extensibilidade do módulo.

4.3.1

PostgreSQL

Na implementação para o *PostgreSQL* foram consideradas dois tipos de ação de sintonia fina: criação de índices e visões materializadas, ambas estruturas de acesso que são criadas de forma hipotética.

4.3.1.1

Índices Hipotéticos

Para a criação dos índices hipotéticos foi utilizado a extensão *HypoPG*, extensão presente em diversos trabalhos que abordam o uso de índices hipotéticos em *PostgreSQL*[33, 34].

Índices criados através dessa extensão são considerados pelo SGBD quando uma consulta é executada com a cláusula *EXPLAIN*, retornando o plano de execução da consulta com o custo estimado caso o índice fosse utilizado.

```
1 select *
2 from hypopg_create_index('create_index_on_hypo_(id)');
```

Consulta 4.6: Consulta para criação de um índice hipotético

A consulta 4.6 apresenta um exemplo de criação de um índice hipotético. O retorno da chamada da função *hypopg_create_index* é um identificador do índice que é utilizado posteriormente para removê-lo.

Para o cálculo do custo de criação e atualização do índice, utiliza-se as fórmulas definidas por Marcos Salles em sua dissertação de mestrado[3], e aqui brevemente resumidas em suas partes principais.

Podemos definir a aproximação para o custo de atualização de um índice como:

$$CA_i = 2 \left[\frac{r}{R} \right] P + cr = r \left(\frac{2P}{R} + c \right)$$

onde

r : número de registros atualizados por um comando

R : número de registros da tabela

P : número de páginas da tabela

c : coeficiente que relaciona o custo de I/O com o custo de CPU para processar um registro em memória. Possui um valor padrão de 1%

```
1 select $2*(2*relpages/reltuples+0.01)
2 from pg_class
3 where relname = $1;
```

Consulta 4.7: Consulta para o cálculo do custo de atualização de um índice

O custo de atualização pode ser calculado com o resultado da consulta 4.7, onde o parâmetro \$1 (linha 3) é o nome da tabela na qual o índice seria criado, e o parâmetro \$2 é o número esperados de registros atualizados.

Podemos definir a aproximação para o custo de criação de um índice como:

$$CC_i = 2P + cR \log R$$

Os parâmetros possuem o mesmo significado definido para a fórmula da estimativa do custo de atualização de um índice.

```

1 select 2*relpages+0.01*reltuples*log(reltuples)
2   from pg_class
3  where relname = $1;
```

Consulta 4.8: Consulta para o cálculo do custo de manutenção de um índice

O custo pode ser calculado com a consulta 4.8 onde o parâmetro \$1 (linha 3) é o nome da tabela na qual o índice seria criado.

4.3.1.2

Visões Materializadas Hipotéticas

Para a criação das visões materializadas hipotéticas foi adotada uma estratégia de reescrita de consulta que permite resgatar o custo estimado da consulta, caso a visão seja criada, e da criação da visão em si em um mesmo plano de execução. De forma a ilustrar esse processo, vamos considerar a Consulta 4.9.

```

1 select l_orderkey,
2       sum(l_extendedprice + (1 - l_discount)) as revenue,
3       o_orderdate,
4       o_shippriority
5 from customer, orders, lineitem
6 where c_mktsegment = 'BUILDING'
7   and c_custkey = o_custkey
8   and l_orderkey = o_orderkey
9   and o_orderdate < date '1995-03-15'
10  and l_shipdate > date '1995-03-15'
11 group by l_orderkey, o_orderdate, o_shippriority
12 order by revenue desc, o_orderdate;
```

Consulta 4.9: Consulta Original

E a visão materializada proposta na Consulta 4.10.

```
1 create materialized view mv_revenue as
2 select l_orderkey,
3       sum(l_extendedprice + (1 - l_discount)) as revenue,
4       o_orderdate,
5       o_shippriority,
6       c_mktsegment,
7       l_shipdate
8 from customer, orders, lineitem
9 where c_custkey = o_custkey and
10      l_orderkey = o_orderkey
11 group by l_orderkey, o_orderdate, o_shippriority,
12          c_mktsegment, l_shipdate
13 order by revenue desc, o_orderdate;
```

Consulta 4.10: Visão Materializada Sugerida

Primeiro buscamos quais condicionais estão presentes na consulta original que não aparecem na visão materializada. Para o exemplo são as condições das linhas 6, 9 e 10:

```
1 where c_mktsegment = 'BUILDING'
2 and o_orderdate < date '1995-03-15'
3 and l_shipdate > date '1995-03-15'
```

Consulta 4.11: Condicionais Extras

Em seguida é montada uma *Common Table Expression* (CTE) que representa a visão materializada hipotética e, por fim, é aplicada uma consulta na CTE com as condicionais extraídas, conforme demonstrado na Consulta 4.12.

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

Consulta 4.12: CTE Resultante

Ao executarmos a Consulta 4.12 com a cláusula *Explain*, é possível extrair o custo de execução da CTE (linhas 1-14) e da nova consulta (linhas 15-19) de forma separada. Esses custos representam, de forma aproximada, o custo de criação da visão materializada, e o custo de execução da consulta na visão materializada.

4.3.2 SQLServer

Na implementação para o *SQLServer* foi considerado apenas a criação de índices hipotéticos. Visões materializadas hipotéticas não foram levadas em consideração devido a limitações da abordagem desenvolvida que impedem seu funcionamento no *SQLServer*.

4.3.2.1 Índices Hipotéticos

O *SQLServer* permite, de forma nativa, que o usuário crie índices hipotéticos. Cabe observar que essa prática depende de comandos não documentados

pela *Microsoft*, sendo necessário a leitura de blogs³⁴ que abordassem o assunto.

```
1 CREATE NONCLUSTERED INDEX [idx_hypothetical]
2 ON [dbo].[table] ( id )
3 WITH STATISTICS_ONLY = -1;
```

Consulta 4.13: Criando um índice hipotético no *SQLServer*

Para criar um índice hipotético é necessária a execução de uma consulta no formato da consulta 4.13. Porém para começar a utilizar o índice para gerar o plano de execução das consultas, é necessário solicitar seu uso ao *SQLServer*.

```
1 SELECT N'DBCC_AUTOPILOT(0,' +
2         CONVERT(nvarchar(MAX), DB_ID()) +
3         N', ' + CONVERT(nvarchar(MAX), object_id) +
4         N', ' + CONVERT(nvarchar(MAX), index_id) +
5         N');'
6 FROM sys.indexes
7 WHERE name = idx_hypothetical;
```

Consulta 4.14: Habilitando os índices hipotéticos

Para cada índice criado é necessário executar um comando *DBCC AUTOPILOT* com informações sobre como usar o índice. Este comando comunica ao *SQLServer* que, ao montar o plano de execução esperado das consultas, que os índices hipotéticos devem ser levados em consideração. A consulta 4.14 gera, e executa, o comando para todos os índices hipotéticos criados.

```
1 DBCC AUTOPILOT(0,5,901578250,5);
```

Consulta 4.15: Exemplo de comando para habilitar índice hipotético

A consulta 4.15 apresenta um exemplo de um comando gerado pela consulta 4.14. Em seguida é necessário ativar o *AUTOPILOT* para então executar a consulta e avaliar seu plano de execução e se o índice reduziu seu custo. O *AUTOPILOT* é um comando não documentado pela *Microsoft* que determina se o SGBD deve, ou não, utilizar os índices hipotéticos para montar os planos de execução.

³<https://www.madeiradata.com/post/performance-tuning-like-a-pro-with-hypothetical-indexes>

⁴<https://github.com/MadeiraData/MadeiraToolbox/blob/master/Utility%20Scripts/Hypothetical%20Indexes%20-%20Example%20Usage.sql>

4.4

Exemplo de execução do Algoritmo SASF

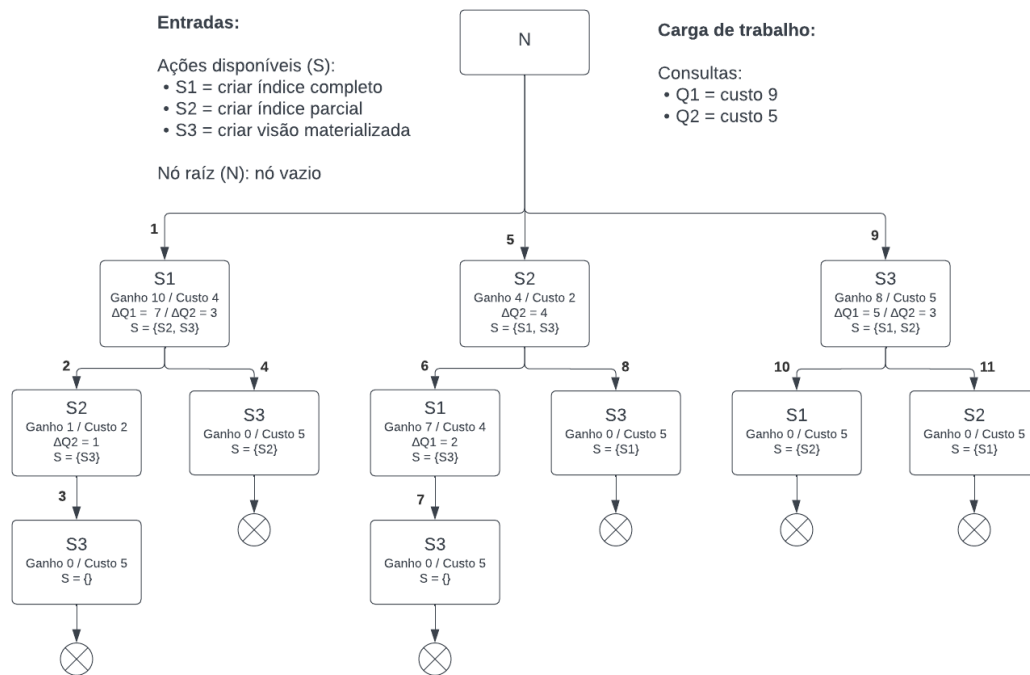


Figura 4.1: Exemplo de execução do Algoritmo 2

Um exemplo de execução do Algoritmo 2 é apresentado na Figura 4.1. O exemplo usa um caso reduzido, com apenas 3 ações de sintonia fina e 2 consultas. para poder demonstrar o funcionamento do algoritmo. A carga de trabalho avaliada é composta por duas consultas, Q1 e Q2, que possuem um custo inicial de 9 e 5 respectivamente. As ações de sintonia fina disponíveis, representadas por S , são 3:

- S1 cria um índice completo que beneficia as consultas Q1 e Q2
- S2 cria um índice parcial que beneficia apenas a consulta Q2
- S3 cria uma visão materializada que beneficia as consultas Q1 e Q2 (reescrevendo as mesmas)

Na **primeira iteração**, a ação S1 é aplicada e, como resultado, temos um ganho global na carga de trabalho de 10. O seu custo estimado é de 4 e temos uma redução de custo nas consultas Q1 e Q2 nos valores de 7 e 3 respectivamente. Como o ganho foi maior do que 0, continuamos descendo na árvore tendo as ações S2 e S3 disponíveis.

Na **segunda iteração**, a ação S2 é aplicada e, como resultado, temos um ganho global na carga de trabalho de 1. O seu custo estimado é de 2 e

temos uma redução de custo na consulta Q2 de 1. Como o ganho foi maior do que 0, continuamos descendo na árvore tendo apenas a ação S3 disponível.

Na **terceira iteração**, a ação S3 é aplicada e, como resultado, temos um ganho global na carga de trabalho de 0. Como não temos mais ações disponíveis, desfazemos as ações S3 e S2, e subimos um nível na árvore.

Na **quarta iteração**, a ação S3 é aplicada e, como resultado, temos um ganho global na carga de trabalho de 0. Como seu ganho foi 0, S2 não é aplicada e subimos um nível na árvore, desfazendo as ações S3 e S1.

Na **quinta iteração**, a ação S2 é aplicada e, como resultado, temos um ganho global na carga de trabalho de 4. O seu custo estimado é de 2 e temos uma redução de custo na consulta Q2 de 4. Como o ganho foi maior do que 0, continuamos descendo na árvore tendo as ações S1 e S3 disponíveis.

Na **sexta iteração**, a ação S1 é aplicada e, como resultado, temos um ganho global na carga de trabalho de 7. O seu custo estimado é de 4 e temos uma redução de custo na consulta Q1 de 7. Como o ganho foi maior do que 0, continuamos descendo na árvore tendo apenas a ação S3 disponível.

Na **sétima iteração**, a ação S3 é aplicada e, como resultado, temos um ganho global na carga de trabalho de 0. Como não temos mais ações disponíveis, desfazemos as ações S3 e S2, e subimos um nível na árvore.

Na **oitava iteração**, a ação S3 é aplicada e, como resultado, temos um ganho global na carga de trabalho de 0. Como seu ganho foi 0, S1 não é aplicada e subimos um nível na árvore, desfazendo as ações S3 e S2.

Na **nona iteração**, a ação S3 é aplicada e, como resultado, temos um ganho global na carga de trabalho de 8. O seu custo estimado é de 5 e temos uma redução de custo nas consultas Q1 e Q2 nos valores de 5 e 3 respectivamente. Como o ganho foi maior do que 0, continuamos descendo na árvore tendo as ações S1 e S2 disponíveis.

Na **décima iteração**, a ação S1 é aplicada e, como resultado, temos um ganho global na carga de trabalho de 0. Como seu ganho foi 0, S2 não é aplicada e subimos um nível na árvore, desfazendo a ação S1.

Na **décima primeira iteração**, a ação S2 é aplicada e, como resultado, temos um ganho global na carga de trabalho de 0. Como seu ganho foi 0, S1 não é aplicada e subimos um nível na árvore, desfazendo as ações S2 e S3.

Após a décima primeira iteração, não existem mais ações disponíveis para serem executadas, chegando ao fim do algoritmo. Nesse momento N é retornado e temos todas as combinações viáveis de sintonia fina para as ações de entrada.

4.5
Interface

Para apresentar as decisões da *TuningChef* para o DBA, foi desenvolvido um sistema *web-based*, utilizando a plataforma *low-code* *Budibase*⁵. Para o desenvolvimento dessa interface, foram listados os requisitos:

- o DBA deve conseguir ver o impacto de uma ação de sintonia fina em toda carga de trabalho
- a evolução do custo total da carga de trabalho deve ser atualizada conforme as ações são escolhidas
- deve ser claro qual é a recomendação da *TuningChef* dentre as opções disponíveis
- o DBA deve poder escolher as soluções que achar melhor, ignorando as recomendações, e ter uma visão do impacto na carga de trabalho de suas escolhas
- o DBA deve ter uma visão clara das consultas que mais consomem recursos do SGBD

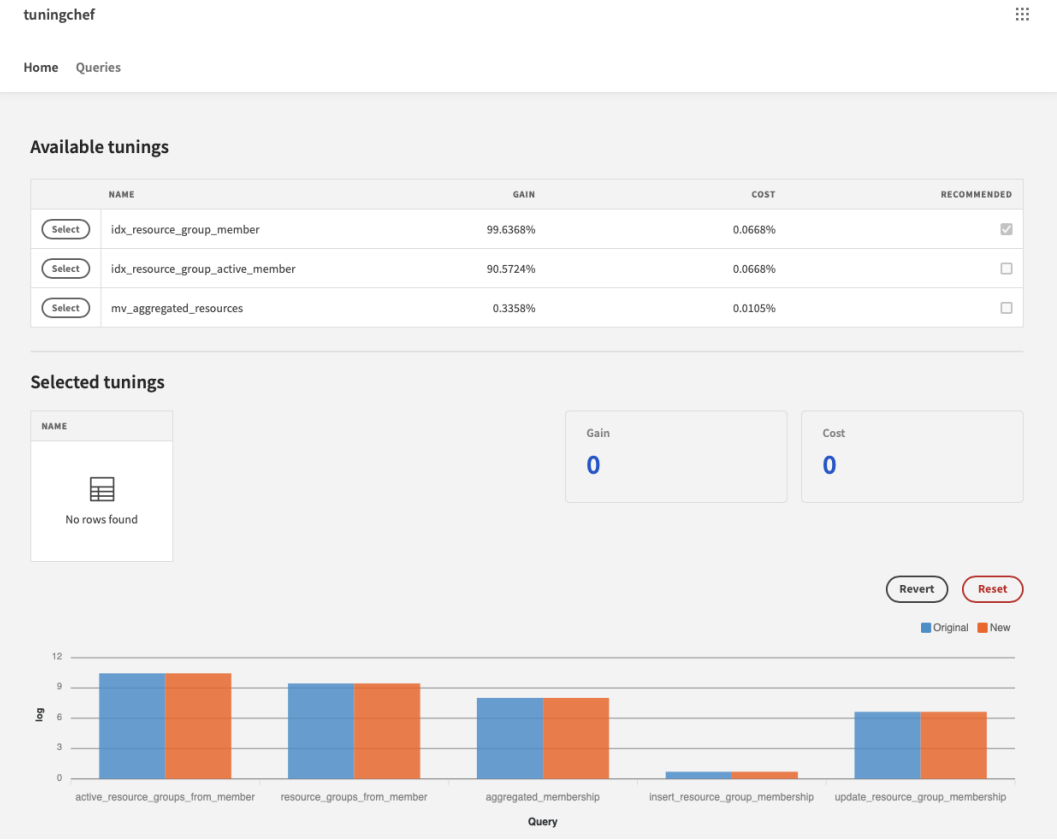


Figura 4.2: Tela inicial da *TuningChef*

⁵<https://budibase.com/>

Na figura 4.2 temos a tela inicial da ferramenta. Nessa tela o DBA tem acesso a lista de ações de sintonia fina disponíveis onde, para cada uma delas, é informado seu nome, ganho, custo e se ela é recomendada para execução. Os valores de ganho e custo são representados como porcentagem (%) do custo total da carga de trabalho que é calculada conforme fórmula apresentada na seção 2.1, facilitando o entendimento do impacto esperado de uma consulta na carga avaliada.

Além das ações de sintonia fina, também são apresentadas quais ações já foram selecionadas até então, ganho e custo global (Figura 4.5) e um gráfico com o estado atual das consultas. O gráfico apresenta duas barras por consulta: uma azul que representa o custo original e um laranja que representa o custo estimado após aplicar as ações de sintonia fina sugeridas. Os gráficos são apresentados em escala logarítmica como forma de facilitar a visualização quando temos valores muito discrepantes dos demais.

Para cada uma das ações de sintonia fina apresentadas na Figura 4.2 é possível obter mais detalhes, os quais são apresentados conforme Figura 4.4. Nessa tela temos algumas informações básicas como nome, tipo, ganho, custo, delta (ganho - custo) e o *SQL*. Além disso temos um gráfico que apresenta o resultado esperado caso aquela ação seja escolhida. Em azul temos o custo atual das consultas, e em laranja o custo esperado caso a ação seja aplicada.

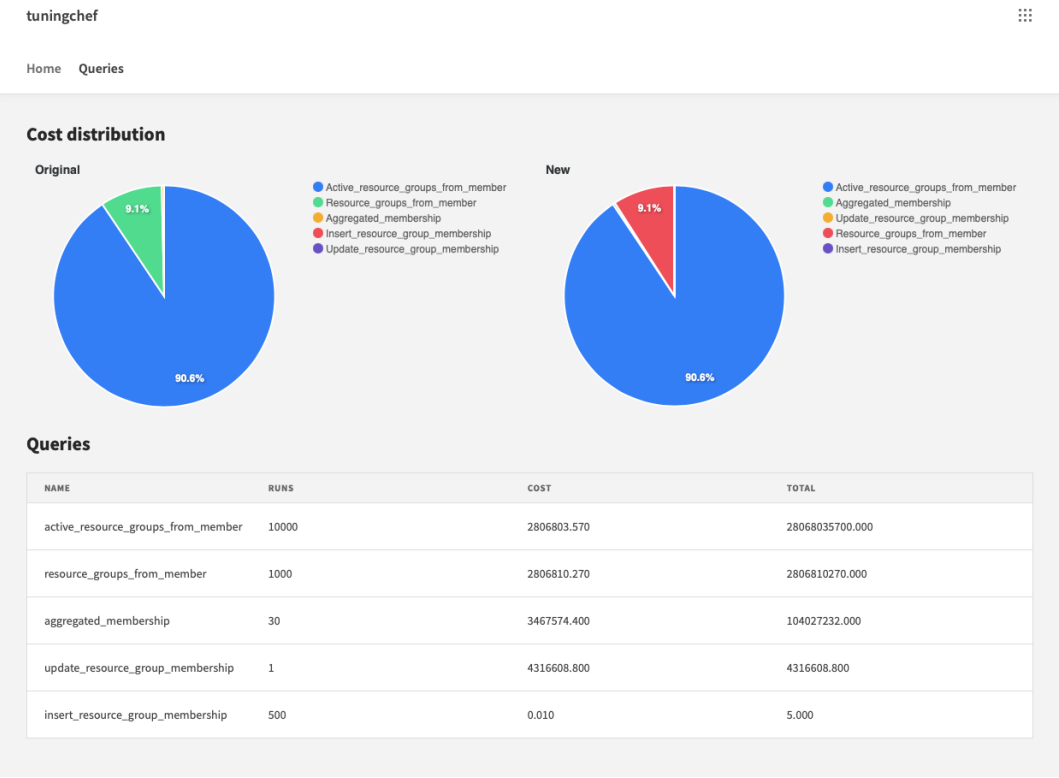


Figura 4.3: Consultas presentes na carga de trabalho

Na figura 4.3 temos a segundo página principal da ferramenta. Nesta página são apresentadas todas as consultas presentes na carga de trabalho analisada, justamente com algumas estatísticas como:

- número de execuções da consulta, representado pelo *runs* na tabela
- o custo de uma única execução, representado pelo *cost* na tabela
- o custo total de execução (custo de uma execução vezes número de execuções), representado pelo *total* na tabela
- a distribuição de peso das consultas pré sintonia fina (gráfico da esquerda)
- a distribuição de peso das consultas após seleção das ações de sintonia fina (gráfico da direita)

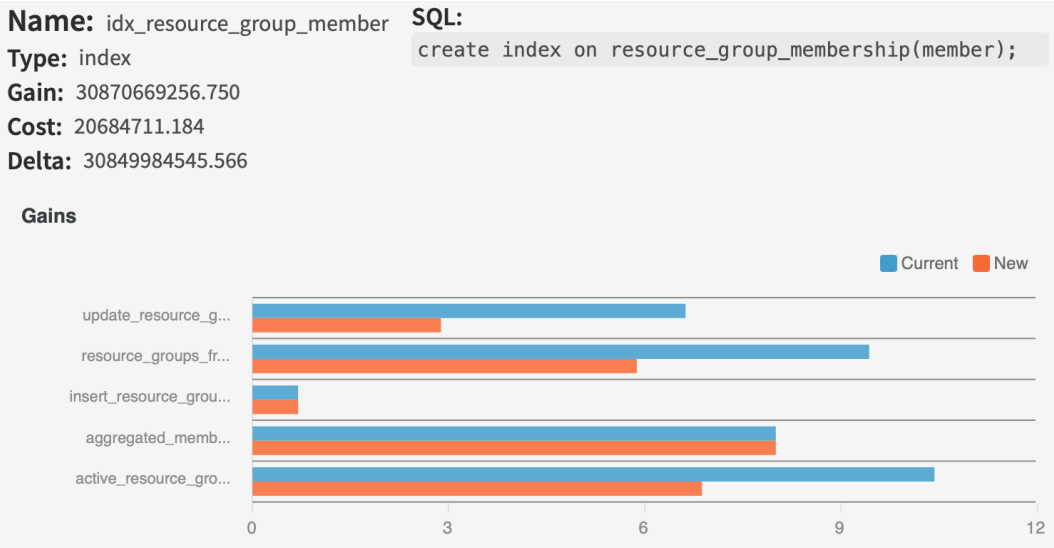


Figura 4.4: Detalhamento da ação

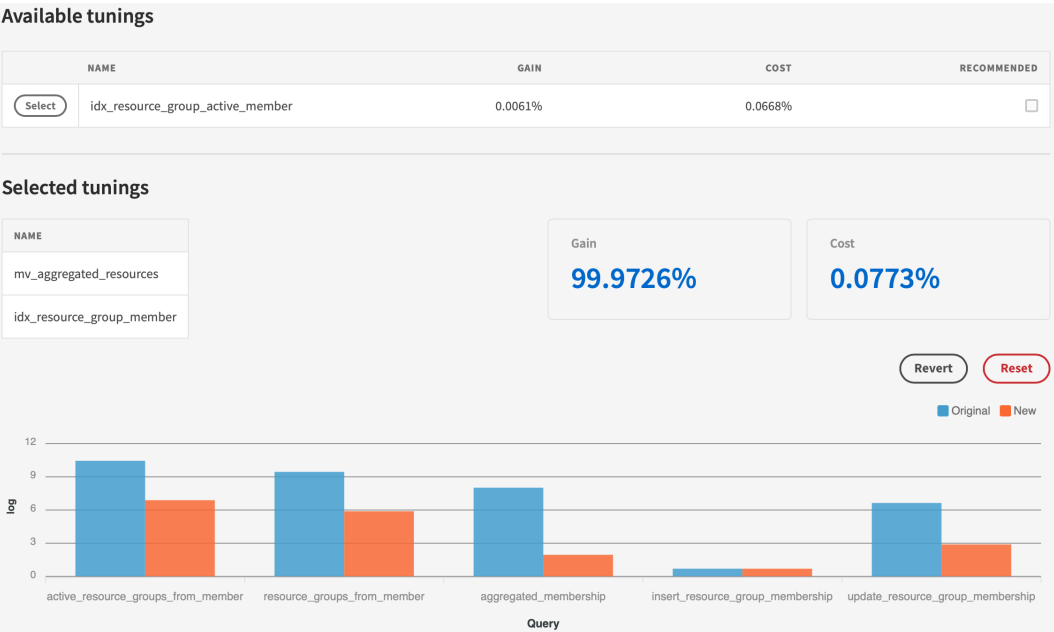


Figura 4.5: Ações escolhidas

Uma vez que as ações de sintonia fina foram escolhidas, uma tela como a da figura 4.5 é apresentada. Nela temos as ações de sintonia fina ainda disponíveis, uma lista das ações escolhidas, o ganho total e custo total esperado das escolhas feitas. Os valores de custo e ganho totais são apresentados como % do custo total da carga de trabalho. Isso significa que, para o exemplo apresentado, temos uma redução esperada de aproximadamente 99% do custo total, em troca de um aumento de aproximadamente 0.08% do custo total.

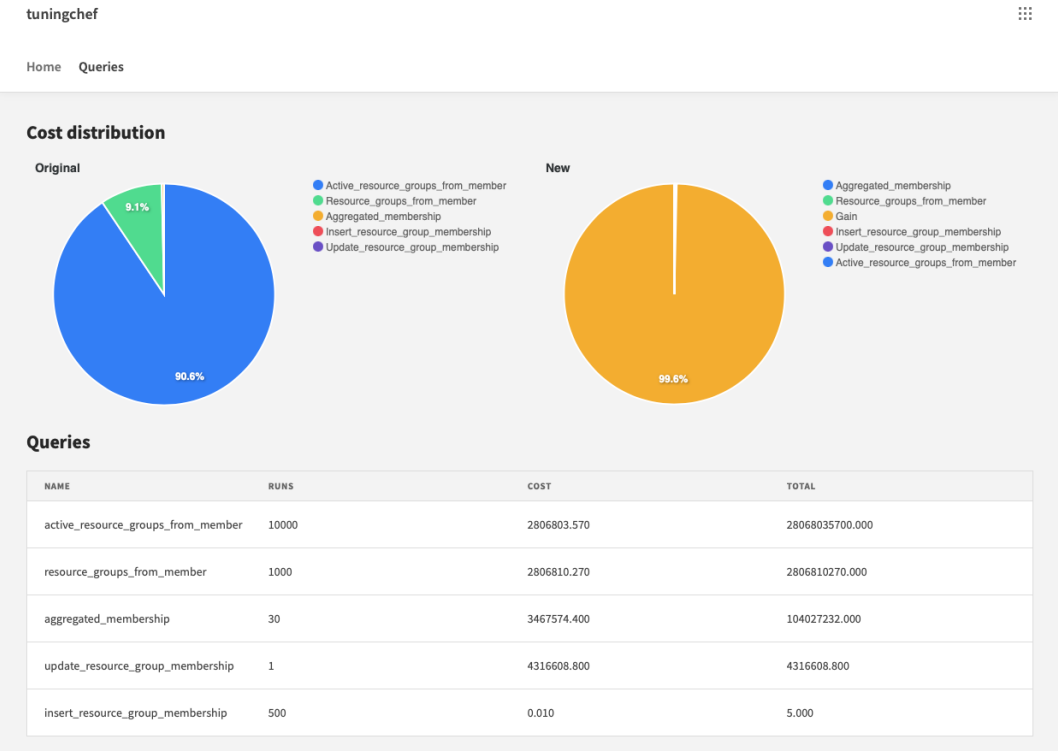


Figura 4.6: Consultas atualizadas

A escolha das ações de sintonia fina também impacta a página de listagem das consultas. Conforme as ações são selecionadas, um novo item pode ser encontrado no gráfico de distribuição de pesos das consultas, conforme observado no gráfico a direita na figura 4.6. Nele temos um novo item chamado *Gain*, que representa o ganho acumulado esperado das ações de sintonia escolhidas. Na figura 4.6 temos este item com um valor de 99.6%, o que representa uma redução de custo total da carga de trabalho de mesmo valor.

4.6
Conclusão

Este capítulo apresentou o processo de desenvolvimento da *TuningChef*, com detalhamento técnico de tecnologias utilizadas e algoritmos desenvolvidos. Também foi apresentada uma abordagem original para a criação de visões materializadas hipotéticas.

5

Resultados experimentais

Nesse capítulo abordaremos o ambiente de teste, com a carga de trabalho e as ações de sintonia fina utilizadas no experimento, e os resultados experimentais em si.

5.1

Ambiente de teste

Para este experimento foi utilizado o *PostgreSQL*, versão 14.4, rodando em um computador isolado com o sistema operacional *Manjaro*, com processador Ryzen 3900x, 16GB de RAM DDR4 2600MHz e 256GB de SSD Sata. O banco foi pré-populado com 10GB de dados do benchmark TPCB. Os dados foram gerados usando o *tcph-kit*¹. O ambiente onde a *TuningChef* foi executado não tem influência nos resultados apresentados, porém a mesma foi executada em uma máquina diferente da do SGBD, de forma a não competirem por recursos.

Como entradas para o avaliador, foram utilizadas as consultas e ações de sintonia fina disponíveis no Capítulo 4 (Estudo de Caso) da *OnDBTuning*[36], com adição de 1 índice e 2 consultas de forma a demonstrar um resultado que misture ações de sintonia fina provenientes de diferentes fontes. Foi escolhido usar as mesmas consultas e ações de sintonia fina que o trabalho [36] por serem pesquisas complementares do grupo BioBD, onde a *OnDBTuning* é responsável por sugerir o que pode ser feito, e a *TuningChef* apoia na decisão do que é mais benéfico de ser feito.

No total, são 8 consultas SQL na carga de trabalho e 57 ações de sintonia fina, divididas em:

- 9 índices simples
- 31 índices compostos
- 16 índices parciais
- 1 visão materializada

Todas as consultas e as ações de sintonia fina encontram-se nos anexos deste trabalho.

¹<https://github.com/gregrahn/tpch-kit>

5.2

Limitações

Por questões de tempo, os resultados experimentais apresentam algumas limitações em sua execução. Os testes foram realizados apenas no *PostgreSQL*, não sendo validado os resultados no *SQLServer*. A utilização da ferramenta também não foi validada por um grupo de DBAs de teste, sendo apenas apresentada pelos alunos do grupo de pesquisa do BioBD.

5.3

Análise dos resultados

Após executada a análise, apenas 8 ações de sintonia fina foram consideradas benéficas, por terem um ganho esperado maior do que o custo, sendo todas sugestões da *OnDBTuning*. A interface web apresenta as opções conforme apresentado na imagem abaixo:

Available tunings

	NAME	GAIN	COST	RECOMMENDED
Select	hyp_p_type_p_partkey	0.0508%	0.0061%	<input checked="" type="checkbox"/>
Select	hyp_l_shipdate	52.1463%	0.2022%	<input type="checkbox"/>
Select	hyp_l_shipdate_l_orderkey	52.1235%	0.2022%	<input type="checkbox"/>
Select	mv_aggregated_resources	6.8482%	0.8785%	<input type="checkbox"/>
Select	hyp_p_part_type_eas	0.0511%	0.0061%	<input type="checkbox"/>
Select	hyp_p_type	0.0510%	0.0061%	<input type="checkbox"/>
Select	hyp_l_partkey	0.0009%	0.2022%	<input type="checkbox"/>
Select	hyp_l_partkey_l_orderkey	0.0003%	0.2022%	<input type="checkbox"/>

Figura 5.1: Ações de sintonia fina disponíveis

E a distribuição de peso das consultas na carga de trabalho:

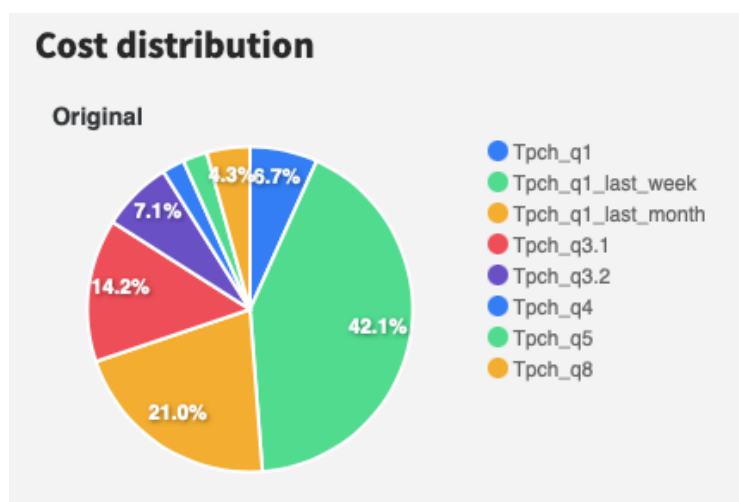


Figura 5.2: Peso das consultas

Como recomendação da TuningChef, temos a criação de 4 índices e 1 visão materializada, sendo eles:

1. índice `hyp_p_type_p_partkey`
2. índice `hyp_p_part_type_eas`
3. índice `hyp_l_shipdate`
4. índice `hyp_p_type`
5. visão materializada `mv_aggregated_resources`

Caso a DBA opte por seguir as recomendações, o seguinte resultado é apresentado:

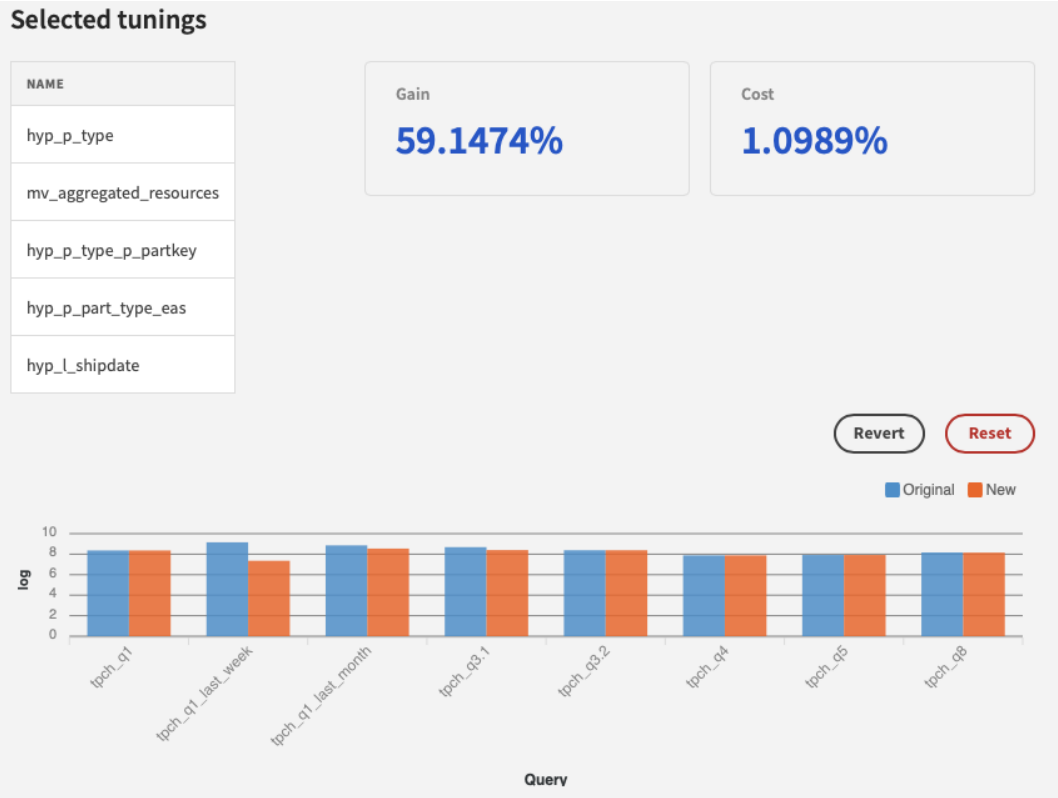


Figura 5.3: Ações de sintonia fina recomendadas aplicadas

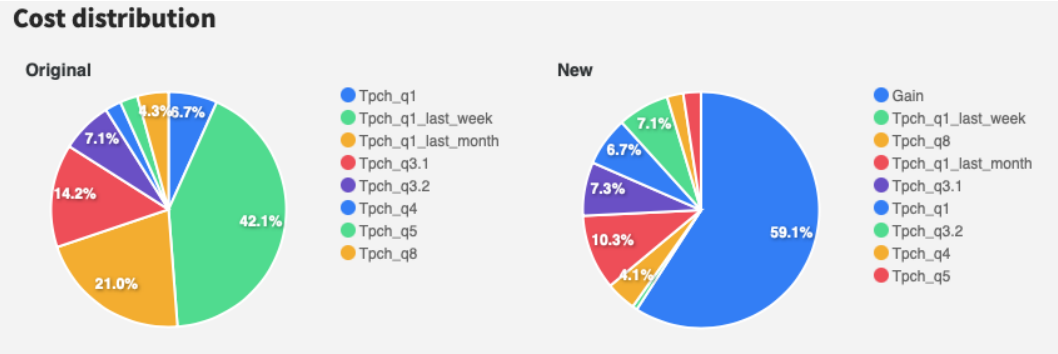


Figura 5.4: Consultas após recomendação aplicada

Apesar de no gráfico da figura 5.3 os ganhos parecerem pequenos, importante ressaltar que os valores são apresentados em escala logarítmica e as pequenas variações representam, na verdade, um ganho de quase 60% na carga de trabalho, em troca de um aumento do custo esperado do banco de aproximadamente 1%. Na figura 5.4 esse comportamento fica mais claro, com o gráfico da direita tendo o item *Gain* com um valor de 59.1%.

Sendo um dos objetivos do trabalho dar o poder de escolha ao DBA, importante ressaltar como esse papel é importante para o teste apresentado. Existem dois índices utilizados que podem ser considerados como conflitantes para a carga de trabalho, por trazerem ganhos similares para o mesmo conjunto de consultas da carga de trabalho, sendo eles:

- *hyp_l_shipdate_l_orderkey*: índice composto nas colunas *l_shipdate* e *l_orderdate* da tabela *lineitem*
- *hyp_l_shipdate*: índice na coluna *l_shipdate* da tabela *lineitem*

Nas ações sugeridas pela *TuningChef*, foi recomendado o *hyp_l_shipdate* por uma diferença em seu ganho de, aproximadamente, 0,02%. Isso acontece porque o índice composto atende os mesmos casos que o índice simples, porém com uma eficiência menor. Entretanto, caso o DBA saiba que uma nova consulta vai ser inserida na carga de trabalho, a qual pode se beneficiar do índice composto, ele pode optar por criar o índice composto e não o simples. Nesse caso, a *TuningChef* mostra o novo resultado esperado, bem como o delta que ele ganharia caso criasse os dois índices:

Available tunings				
	NAME	GAIN	COST	RECOMMENDED
Select	mv_aggregated_resources	6.8482%	0.8785%	<input type="checkbox"/>
Select	hyp_p_part_type_eas	0.0511%	0.0061%	<input type="checkbox"/>
Select	hyp_p_type	0.0510%	0.0061%	<input type="checkbox"/>
Select	hyp_p_type_p_partkey	0.0508%	0.0061%	<input type="checkbox"/>
Select	hyp_l_shipdate	0.0228%	0.2022%	<input type="checkbox"/>
Select	hyp_l_partkey	0.0009%	0.2022%	<input type="checkbox"/>
Select	hyp_l_partkey_l_orderkey	0.0003%	0.2022%	<input type="checkbox"/>

Selected tunings		
NAME	Gain	Cost
hyp_l_shipdate_l_orderkey	52.1235%	0.2022%

Figura 5.5: Resultado da intervenção manual

Na figura 5.5 temos o resultado esperado caso o índice composto seja criado, bem como o resultado esperado caso o índice simples seja criado logo

em seguida, com seu ganho reduzido. Originalmente o ganho do índice simples era de aproximadamente 52,15% e, na figura, ele se encontra com um ganho aproximado de 0,02% apenas, valor menor do que seu custo de 0,2%, tornando seu delta negativo e não justificando sua criação.

5.4

Conclusão

Este capítulo apresentou resultados experimentais que buscaram detalhar como os requisitos descritos no capítulo 3 foram atingidos. Foram usadas ações de sintonia fina de diferentes fontes, sendo elas a *OnDBTuning* e de um DBA. Também foi apresentado como o administrador pode intervir no processo e fazer suas próprias escolhas caso necessário, e como isso pode ser benéfico para o processo de sintonia fina.

6

Conclusão

Neste trabalho de pesquisa foi projetada e implementada uma ferramenta para apoiar um DBA no momento de escolha de quais ações de sintonia fina devem ser executadas. De forma a demonstrar a extensibilidade da ferramenta, foram implementadas duas estratégias de avaliação de sintonia fina para os casos de criação de índices e de visões materializadas. A estratégia de índices foi implementada para dois SGBDs para demonstrar que a *TuningChef* não está restrita a um único sistema de banco de dados. Resultados parciais desse trabalho foram publicados antes da defesa desta dissertação [37].

6.1

Contribuições da pesquisa

Como principal contribuição desta pesquisa, temos uma nova abordagem arquitetural para o processo de sintonia fina, a qual é flexível o suficiente para integrar ferramentas existentes com conhecimentos adquiridos do DBA, oferecendo uma solução sistemática para a tomada de decisão, o que até então não existia. Para demonstrar a viabilidade dessa arquitetura, temos a implementação da ferramenta *TuningChef*, a qual o DBA pode usar como apoio para tomar melhores decisões na hora de realizar o *tuning* de seu banco de dados relacional, capaz de recomendar as soluções de maior custo-benefício, mas que também dá liberdade ao administrador para montar o seu próprio plano de ação e ter uma visão completa do impacto esperado na carga de trabalho.

Quando comparado com as soluções descritas na seção 2.6, a *TuningChef* traz uma visualização do impacto das ações de sintonia fina de uma forma global na carga de trabalho, visão a qual é inexistente nos trabalhos apresentados. Existem ferramentas comerciais (em especial da Oracle) que oferecem uma visualização parecida, porém restrita apenas ao ecossistema de um único SGBD, o que força ao DBA a aprender diferentes ferramentas para o mesmo trabalho, caso seja responsável por administrar diferentes sistemas de bancos de dados. A arquitetura proposta neste trabalho não se restringe a um único SGBD, podendo ser estendida conforme necessário.

Outro ponto de destaque na comparação é o papel do DBA na ferramenta.

Na *TuningChef* ele pode optar por ignorar totalmente as recomendações da ferramenta e fazer as próprias escolhas, possivelmente devido a algum conhecimento futuro de mudança na carga de trabalho. A maioria das soluções descritas na seção 2.6, no entanto, engessam a participação do DBA em escolhas de tudo ou nada, ou em casos mais flexíveis, permitem que o mesmo tome decisões binárias de "sim" ou "não" para cada ação de sintonia fina, sem permitir que o mesmo monte a própria combinação.

Como contribuição adicional temos uma solução original para avaliar o custo e o benefício de uma visão materializada hipotética. Essa solução foi desenvolvida para permitir demonstrar a extensibilidade da *TuningChef*, demonstrando que estratégias podem ser desenvolvidas para diferentes tipos de ações de sintonia fina. A maior vantagem dessa solução é a capacidade de utilizar estatísticas fornecidas pelo SGBD do banco para avaliar o impacto esperado da visão materializada.

6.2

Trabalhos futuros

Devido a uma limitação de tempo, as seguintes atividades não foram realizadas:

1. Integração direta com a *OnDBTuning*. A integração foi manual nos resultados experimentais.
2. Adicionar limitações de recursos, como memória disponível, como critério para o processo de decisão.

Apesar de não realizadas, essas atividades não comprometem os resultados apresentados por não invalidarem a arquitetura e algoritmos propostos.

Trabalhos futuros podem contemplar atividades como:

1. Integração direta com a *OnDBTuning* ou outras ferramentas de sugestão de ações de sintonia fina: para este trabalho, a integração foi feita de forma manual, obrigando o operador a rodar as duas ferramentas de forma isolada e, manualmente, transformar a saída da *OnDBTuning* no formato esperado pela *TuningChef*. A integração direta remove a etapa manual executada.
2. Desenvolvimento de novas heurísticas de sintonia fina para avaliação: neste trabalho focamos em índices e visões materializadas hipotéticas, mas novas heurísticas podem ser desenvolvidas para avaliar outras ações de sintonia fina, como *Column Tetris*, que visa reduzir o consumo de disco e I/O ordenando as colunas de uma tabela de forma ótima[35].

3. Criação de testes automatizados no código da ferramenta: testes automatizados auxiliam na hora de garantir a corretude do *software*. Apesar de não encontrados durante a execução dos testes para esse trabalho, alguns casos de borda podem apresentar problemas.
4. Otimização do algoritmo do módulo de avaliação: a exploração das combinações das ações de sintonia fina pode ser otimizada com estratégias mais agressivas de *Branch and Bound* e algoritmos de busca em grafo, evitando percorrer o mesmo caminho mais de uma vez onde cabível.
5. Troca da ferramenta *low-code* utilizada na interface (*Budibase*) por uma interface web feita com tecnologias mais flexíveis como *React*, *Vue*, *Svelte* ou *Angular* (exemplos de frameworks web para *javascript*).
6. Realizar pesquisas qualitativas sobre a qualidade da *TuningChef* com DBAs. Não foi feita nenhuma pesquisa sobre o uso da ferramenta com um grupo de testes. A pesquisa é necessária para avaliar o seu uso em casos de mundo real, fora dos testes utilizados neste trabalho.

Referências bibliográficas

- [1] RAMAKRISHNAN, R.; GEHRKE, J.. **Sistemas de Gerenciamento de Bancos de Dados**. AMGH, 3rd edition, 2008.
- [2] SHASHA, D.; BONNET, P.. **Database Tuning: Principles, Experiments and Troubleshooting Techniques**. Morgan Kaufmann, 1st edition, 2002.
- [3] SALLES, M. A. V.. **Criação autônoma de Índices em bancos de dados**. Dissertação de mestrado, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2004.
- [4] ELMASRI R.; NAVATHE, S. B.. **Sistemas de Banco de Dados**. Pearson Universidades, 7th edition, 2019.
- [5] MONTEIRO, J. M.; LIFSCHITZ, S.. **Estado da arte em auto-sintonia do projeto físico de banco de dados**. Monografias em Ciência da Computação nº 41/08, Pontifícia Universidade Católica do Rio de Janeiro, 2008.
- [6] GUTTMAN, A.. **R-trees: A dynamic index structure for spatial searching**. ACM SIGMOD Record, 14:47–57, 1984.
- [7] STONEBRAKER, M.. **The case for partial indexes**. ACM SIGMOD Record, 18(4):4–11, Dec 1989.
- [8] GUPTA, M.; BADAL, D.. **A study on indexes and index structures**. Intensity: International Journal of Applied Social Science Research, 2:212–222, 02 2013.
- [9] BRUNO, N.. **Automated Physical Database Design and Tuning**. CRC Press, 1st edition, 2017.
- [10] SILBERSCHATZ, A.. **Sistema de banco de dados**. GEN LTC, 6th edition, 2012.
- [11] CHIRKOVA, R.; YANG, J.. **Materialized Views**. Now Publishers, 4th edition, 2012.

- [12] COSTA, R. L. D. C.; LIFSCHITZ, S.; SALLES, M. A. V. **Index self-tuning with agent-based databases**. CLEI Electronic Journal, 6(1), Sep 2018.
- [13] FUENTES, A. D.. **Sintonia fina automática com índices parciais**. Dissertação de mestrado, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Mar 2016.
- [14] FUENTES, A. D.; ALMEIDA, A. C.; BRAGANHOLO, V.; LIFSCHITZ, S.. **Database tuning with partial indexes**. In: SIMPOSIO BRASILEIRO DE BANCO DE DADOS - 33RD SBBB, p. 181–192, Brasil, 2018.
- [15] MORELLI, E. M. T.. **Recriação automática de índices em um sgbd relacional**. Dissertação de mestrado, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Sep 2006.
- [16] CARVALHO, A. W.. **Criação automática de visões materializadas em sgbds relacionais**. Dissertação de mestrado, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Mar 2011.
- [17] ALMEIDA, A. C. B.. **Framework para apoiar a sintonia fina de banco de dados**. Tese de doutorado, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Sep 2013.
- [18] OLIVEIRA, R. P.. **Sintonia fina baseada em ontologia: o caso de visões materializadas**. Dissertação de mestrado, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Mar 2015.
- [19] ALLEN, D. V.; PAVLO A.; GORDON G. J.; ZHANG B.. **Automatic database management system tuning through large-scale machine learning**. In: PROCEEDINGS OF THE 2017 ACM INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, p. 1009–1024, Chicago Illinois USA, May 2017. ACM.
- [20] BEUTEL A.; KRASKA T.; CHI E. H.; DEAN J.; POLYZOTIS N.. **A machine learning approach to databases indexes**. ML Systems Workshop at NIPS 2017, p. 5, 2017.
- [21] ZHANG J.; ZHOU K.; LI G.; LIU Y.; XIE M.; CHENG B.; XING J.. **Cdb-tune+: An efficient deep reinforcement learning-based automatic cloud database tuning system**. The VLDB Journal, 30(6):959–987, Nov 2021.
- [22] DING B.; DAS S.; MARCUS R.; WU W.; CHAUDHURI S.; NARASAYYA V. R.. **Ai meets ai: Leveraging query executions to improve index**

- recommendations. In: PROCEEDINGS OF THE 2019 INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, p. 1241–1258, Amsterdam Netherlands, Jun 2019. ACM.
- [23] GRUBER T.. **Ontologies**. In: PROCEEDINGS OF ENCYCLOPEDIA OF DATABASE SYSTEMS, p. 1959–1959. Springer, 2009.
- [25] MONTEIRO J.. **Uma abordagem não intrusiva para a manutenção automática do projeto físico de banco de dados**. Tese de doutorado, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2008.
- [26] OLIVEIRA, R. P.. **COMBINAÇÃO E SELEÇÃO AUTOMÁTICA DE AÇÕES DE SINTONIA FINA**. Doutor em ciências - informática, PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO, Rio de Janeiro, Brazil, Sep 2019.
- [27] OLIVEIRA, R. P.; BAIÃO F.; ALMEIDA, A. C.; SCHWABE D.; LIFSCHITZ S.. **Outer-tuning: an integration of rules, ontology and rdbms**. In: ANAIS DO XV SIMPÓSIO BRASILEIRO DE SISTEMAS DE INFORMAÇÃO, p. 471–478, Porto Alegre, RS, Brasil, 2019. SBC.
- [28] PERCILIANO, L. S. S.; SANTOS, V.; BAIÃO, F.; HAEUSLER, E. H.; LIFSCHITZ, S.; ALMEIDA, A. C.. **Inferencing relational database tuning actions with ondbtuning ontology**. In: ANAIS DO XXXVI SIMPÓSIO BRASILEIRO DE BANCO DE DADOS (SBBD 2021), p. 157–168, Brasil, Oct 2021. Sociedade Brasileira de Computação - SBC.
- [29] HU, D.; CHEN, Z.; WU, J.; SUN, J.; CHEN, H.. **Persistent memory hash indexes: An experimental evaluation**. Proc. VLDB Endow., 14(5):785–798, jan 2021.
- [30] KHAN, F.; CHEN, B.; VARRO, D.; MCINTOSH, S.. **An empirical study of type-related defects in python projects**. IEEE Transactions on Software Engineering, 48(8):3145–3158, 2022.
- [31] SIEK, J. **Challenges and progress toward efficient gradual typing (invited talk)**. In: PROCEEDINGS OF THE 13TH ACM SIGPLAN INTERNATIONAL SYMPOSIUM ON ON DYNAMIC LANGUAGES, DLS 2017, p. 2, New York, NY, USA, 2017. Association for Computing Machinery.
- [32] CHEN, Z.; LI, Y.; CHEN, B.; MA, W.; CHEN, L.; XU, B.. **An empirical study on dynamic typing related practices in python systems**. In: PROCEEDINGS OF THE 28TH INTERNATIONAL CONFERENCE ON

- PROGRAM COMPREHENSION, ICPC '20, p. 83–93, New York, NY, USA, 2020. Association for Computing Machinery.
- [33] KOSSMANN, J.; HALFPAP, S.; JANKRIFT, M.; SCHLOSSER, R.. **Magic mirror in my hand, which is the best in the land? an experimental evaluation of index selection algorithms.** Proc. VLDB Endow., 13(12):2382–2395, 2020.
- [34] SCHLOSSER, R.; HALFPAP, S.. **A decomposition approach for risk-averse index selection.** In: 32ND INTL CONF SCIENTIFIC AND STATISTICAL DATABASE MANAGEMENT, 2020.
- [35] SOUZA, V.. **Sintonia fina de um banco postgresql: Estudo de caso enem.** Trabalho de conclusão de curso, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Jun 2018.
- [36] PERCILIANO, L.. **Inferência de tuning através da ondbtuning.** Mestre em informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brazil, Dec 2021.
- [37] SOUZA, V.; LIFSCHITZ, S.. **Tuningchef: an approach for choosing the best cost-benefit database tuning actions.** In: ANAIS DO XXXVII SIMPÓSIO BRASILEIRO DE BANCO DE DADOS (SBBBD 2022), Brasil, 2022. Sociedade Brasileira de Computação - SBC.

A

Consultas TPC-H

```
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_discprice,
       sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
       avg(l_quantity) as avg_qty,
       avg(l_extendedprice) as avg_price,
       avg(l_discount) as avg_disc,
       count(*) as count_order
from lineitem
where l_shipdate <= date '1998-12-01' - interval '87 days'
group by l_returnflag, l_linestatus
order by l_returnflag, l_linestatus;
```

Figura A.1: Consulta TPC-H Q1.1

```
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_discprice,
       sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
       avg(l_quantity) as avg_qty,
       avg(l_extendedprice) as avg_price,
       avg(l_discount) as avg_disc,
       count(*) as count_order
from lineitem
where l_shipdate > date '1998-12-01' - interval '1 month'
group by l_returnflag, l_linestatus
order by l_returnflag, l_linestatus;
```

Figura A.2: Consulta TPC-H Q1.2

```

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_discprice,
       sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
       avg(l_quantity) as avg_qty,
       avg(l_extendedprice) as avg_price,
       avg(l_discount) as avg_disc,
       count(*) as count_order
from lineitem
where l_shipdate > date '1998-12-01' - interval '1 week'
group by l_returnflag, l_linestatus
order by l_returnflag, l_linestatus;

```

Figura A.3: Consulta TPC-H Q1.3

Na figura A.1 temos uma instanciação da consulta Q1 do benchmark TPC-H, seguindo as regras descritas em sua especificação¹. Já nas figuras A.2 e A.3 temos instanciações que alteram a cláusula condicional de \leq (menor ou igual) para $>$ (maior), com o objetivo de restringir o espaço de busca para consultas que olhem apenas para entradas mais recentes.

```

select l_orderkey,
       sum(l_extendedprice + (1 - l_discount)) as revenue,
       o_orderdate,
       o_shippriority
from customer, orders, lineitem
where c_mktsegment = 'BUILDING'
   and c_custkey = o_custkey
   and l_orderkey = o_orderkey
   and o_orderdate < date '1995-03-15'
   and l_shipdate > date '1995-03-15'
group by l_orderkey, o_orderdate, o_shippriority
order by revenue desc, o_orderdate;

```

Figura A.4: Consulta TPC-H Q3.1

¹https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v3.0.1.pdf

```
select l_orderkey,  
       sum(l_extendedprice + (1 - l_discount)) as revenue,  
       o_orderdate,  
       o_shippriority  
from customer, orders, lineitem  
where c_mktsegment = 'AUTOMOBILE'  
   and c_custkey = o_custkey  
   and l_orderkey = o_orderkey  
   and o_orderdate < date '1995-03-30'  
   and l_shipdate > date '1995-03-30'  
group by l_orderkey, o_orderdate, o_shippriority  
order by revenue desc, o_orderdate;
```

Figura A.5: Consulta TPC-H Q3.2

Nas figuras A.4 e A.5 temos instâncias da consulta Q3 do benchmark TPC-H.

```
select o_orderpriority,  
       count(*) as order_count  
from orders, lineitem  
where o_orderdate >= date '1993-07-01'  
   and o_orderdate < date '1993-07-01' + interval '3' month  
   and l_orderkey = o_orderkey  
   and l_commitdate < l_receiptdate  
group by o_orderpriority  
order by o_orderpriority;
```

Figura A.6: Consulta TPC-H Q4

Na figura A.6 temos uma instância da consulta Q4 do benchmark TPC-H.

```

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 = 'ASIA'
   and o_orderdate >= date '1994-01-01'
   and o_orderdate < date '1994-01-01' + interval '1' year
group by n_name
order by revenue desc;

```

Figura A.7: Consulta TPC-H Q5

Na figura A.7 temos uma instanciação da consulta Q5 do benchmark TPC-H.

```

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

```

Figura A.8: Consulta TPC-H Q8

Na figura A.8 temos uma instanciação da consulta Q8 do benchmark TPC-H.

B

Ações de sintonia fina

```
create index hyp_c_mktsegment on customer (c_mktsegment);
create index hyp_l_commitdate on lineitem (l_commitdate);
create index hyp_l_partkey on lineitem (l_partkey);
create index hyp_l_receiptdate on lineitem (l_receiptdate);
create index hyp_l_shipdate on lineitem (l_shipdate);
create index hyp_l_suppkey on lineitem (l_suppkey);
create index hyp_o_orderdate on orders (o_orderdate);
create index hyp_p_type on part (p_type);
create index hyp_r_name on region (r_name);
```

Figura B.1: Índices Simples

Na figura B.1 temos os índices simples sugeridos pela *OnDBTuning*.

```

create index hyp_c_custkey_c_mktsegment on customer (c_custkey, c_mktsegment);
create index hyp_c_custkey_c_nationkey on customer(c_custkey, c_nationkey);
create index hyp_c_mktsegment_c_custkey on customer (c_mktsegment, c_custkey);
create index hyp_c_nationkey_c_custkey on customer (c_nationkey, c_custkey);

create index hyp_l_commitdate_l_receiptdate on lineitem (l_commitdate, l_receiptdate);
create index hyp_l_commitdate_l_orderkey on lineitem (l_commitdate, l_orderkey);
create index hyp_l_orderkey_l_commitdate on lineitem (l_orderkey, l_commitdate);
create index hyp_l_orderkey_l_partkey on lineitem (l_orderkey, l_partkey);
create index hyp_l_orderkey_l_receiptdate on lineitem (l_orderkey, l_receiptdate);
create index hyp_l_orderkey_l_shipdate on lineitem (l_orderkey, l_shipdate);
create index hyp_l_orderkey_l_suppkey on lineitem (l_orderkey, l_suppkey);
create index hyp_l_partkey_l_orderkey on lineitem (l_partkey, l_orderkey);
create index hyp_l_receiptdate_l_orderkey on lineitem (l_receiptdate, l_orderkey);
create index hyp_l_receiptdate_l_commitdate on lineitem (l_receiptdate, l_commitdate);
create index hyp_l_shipdate_l_orderkey on lineitem (l_shipdate, l_orderkey);
create index hyp_l_suppkey_l_orderkey on lineitem (l_suppkey, l_orderkey);
create index hyp_l_suppkey_l_partkey on lineitem (l_suppkey, l_partkey);

create index hyp_n_nationkey_n_regionkey on nation (n_nationkey, n_regionkey);
create index hyp_n_regionkey_n_nationkey on nation (n_regionkey, n_nationkey);

create index hyp_o_custkey_o_orderdate on orders (o_custkey, o_orderdate);
create index hyp_o_custkey_o_orderkey on orders (o_custkey, o_orderkey);
create index hyp_o_orderdate_o_custkey on orders (o_orderdate, o_custkey);
create index hyp_o_orderdate_o_orderkey on orders (o_orderdate, o_orderkey);
create index hyp_o_orderkey_o_custkey on orders (o_orderkey, o_custkey);
create index hyp_o_orderkey_o_orderdate on orders (o_orderkey, o_orderdate);

create index hyp_p_partkey_p_type on part (p_partkey, p_type);
create index hyp_p_type_p_partkey on part (p_type, p_partkey);

create index hyp_r_name_r_regionkey on region (r_name, r_regionkey);
create index hyp_r_regionkey_r_name on region (r_regionkey, r_name);

create index hyp_s_nationkey_s_suppkey on supplier (s_nationkey, s_suppkey);
create index hyp_s_suppkey_s_nationkey on supplier (s_suppkey, s_nationkey);

```

Figura B.2: Índices Compostos

Na figura B.2 temos os índices compostos sugeridos pela *OnDBTuning*.


```

create index hyp_c_part_mktsegment_building on customer (c_mktsegment)
  where c_mktsegment = 'BUILDING';
create index hyp_c_part_mktsegment_automobile on customer (c_mktsegment)
  where c_mktsegment = 'AUTOMOBILE';

create index hyp_l_part_shipdate_le87d on lineitem (l_shipdate)
  where l_shipdate <= date '1998-12-01' - interval '87 days';
create index hyp_l_part_shipdate_g1995_03_15 on lineitem (l_shipdate)
  where l_shipdate > date '1995-03-15';

create index hyp_o_part_orderdate_ge1994_01_01 on orders (o_orderdate)
  where o_orderdate >= date '1994-01-01';
create index hyp_o_part_orderdate_ge1993_07_01 on orders (o_orderdate)
  where o_orderdate >= date '1993-07-01';
create index hyp_o_part_orderdate_l1995_03_15 on orders (o_orderdate)
  where o_orderdate < date '1995-03-15';
create index hyp_o_part_orderdate_g1995_03_30 on orders (o_orderdate)
  where o_orderdate > date '1995-03-30';
create index hyp_o_part_orderdate_l1995_03_30 on orders (o_orderdate)
  where o_orderdate < date '1995-03-30';
create index hyp_o_part_orderdate_bt on orders (o_orderdate)
  where o_orderdate between date '1995-01-01' and date '1996-12-31';
create index hyp_o_part_orderdate_l1994_01_01_plus1y on orders (o_orderdate)
  where o_orderdate < date '1995-01-01' + interval '1 year';
create index hyp_o_part_orderdate_l1993_07_01_plus3m on orders (o_orderdate)
  where o_orderdate < date '1993-07-01' + interval '3 months';

create index hyp_p_part_type_eas on part (p_type)
  where p_type = 'ECONOMY ANODIZED STEEL';

create index hyp_r_part_name_asia on region (r_name)
  where r_name = 'ASIA';
create index hyp_r_part_name_america on region (r_name)
  where r_name = 'AMERICA';

create index hyp_l_part_shipdate_g87d on lineitem (l_shipdate)
  where l_shipdate > date '1998-12-01' - interval '87 days';

```

Figura B.3: Índices Parciais

Na figura B.3 temos os índices parciais sugeridos pela *OnDBTuning*, com exceção do último (*hyp_l_part_shipdate_g87d*) que foi adicionado manualmente.

```
create materialized view mv_revenue as
select l_orderkey,
       sum(l_extendedprice + (1 - l_discount)) as revenue,
       o_orderdate,
       o_shippriority,
       c_mktsegment,
       l_shipdate
  from customer, orders, lineitem
 where c_custkey = o_custkey and
       l_orderkey = o_orderkey
 group by l_orderkey, o_orderdate, o_shippriority,
         c_mktsegment, l_shipdate
 order by revenue desc, o_orderdate;
```

Figura B.4: Visão Materializada para as consultas Q3

Na figura B.4 temos uma visão materializada sugerida pela *OnDBTuning* para as consultas A.4 e A.5.