Pontifícia Universidade Católica
do Rio de Janeiro

Alain Domínguez Fuentes

Sintonia fina automática com índices parciais

Dissertação de Mestrado

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

Orientador: Prof. Sérgio Lifschitz

Rio de Janeiro
Março de 2016

Alain Domínguez Fuentes

Sintonia fina automática com índices parciais

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

Sérgio Lifschitz
Orientador
Departamento de Informática – PUC-Rio

Vanessa Braganholo Murta
UFF

Rogério Luís de Carvalho Costa
PUC-Rio

Prof. Márcio da Silveira Carvalho
Coordenador Setorial do Centro
Técnico Científico – PUC-Rio

Rio de Janeiro, 29 de março de 2016

Alain Domínguez Fuentes

Graduado em Ciência da Computação (2010) pela Universidad de La Habana. Atua principalmente nos seguintes temas: banco de dados, sintonia-fina de banco de dados.

Ficha Catalográfica

Domínguez Fuentes, Alain

Sintonia fina automática com índices parciais / Alain Domínguez Fuentes ; orientador: Sérgio Lifschitz. – 2016.
82 f. : il. color. ; 30 cm

Dissertação (mestrado)–Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2016.
Inclui bibliografia

1. Informática – Teses. 2. Bancos de dados. 3. Sintonia fina. 4. Índices parciais. I. Lifschitz, Sérgio. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Agradecimentos

A minha esposa Lizmary Zulueta Lópes, companheira de jornada, incentivadora e alicerce. Meu filho Eddard Domínguez.

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

A minha mãe Maira Fuentes e meu pai Pedro Domínguez, que me ensinaram que a única forma de vencer é estudando.

A todos os amigos que conquistei aqui na PUC-Rio, em especial os amigos: Adriel Gárcia Hernández, Alexander Chávez López, Yoanni Sarmiento Arias, Alejandro Martinez Rivero, Elvismary Molina de Armas, Liester Cruz Castro, Julio Omar Prieto Entenza e Liander Millan.

Minha grande professora Lucina García Hernández.

A Capes, ao CNPq e a PUC-Rio, pelo apoio financeiro concedido ao longo do curso.

A todos os funcionários e professores do departamento de Informática da PUC-Rio, meu agradecimento e admiração.

Resumo

Fuentes, Alain Domínguez; Lifschitz, Sérgio. **Sintonia fina automática com índices parciais**. Rio de Janeiro, 2016. 82p. Dissertação de Mestrado. — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Os índices parciais são estruturas de acesso no nível físico dos bancos de dados que permitem definir um subconjunto das tuplas de uma tabela, através de uma expressão condicional. Nesta dissertação estuda-se a identificação e subsequente criação automática de índices parciais que possam contribuir na melhoria do desempenho de um sistema de banco de dados. É proposto um algoritmo que examina, para cada consulta relevante, os conjuntos de atributos indexáveis para os quais a criação de um índice parcial poderia influenciar o otimizador de consultas na geração de planos mais eficientes. É realizada uma mineração de padrões de atributos indexáveis para se obter atributos correlacionados segundo a frequência das consultas na carga de trabalho considerada. Chega-se a uma proposta para um conjunto de índices parciais candidatos também se considerando uma heurística de benefícios. Realiza-se uma análise de sintonia fina em função da seleção de uma configuração de índices parciais e índices completos. A implementação das técnicas e algoritmos propostos nesta pesquisa é feita no *framework* DBX, que permite instanciar técnicas de sintonia fina local e global para bancos de dados relacionais.

Palavras-chave

bancos de dados; sintonia fina; índices parciais

Abstract

Fuentes, Alain Domínguez; Lifschitz, Sérgio (Advisor). **Database self-tuning with partial indexes**. Rio de Janeiro, 2016. 82p. MSc. Dissertation. — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Partial indexes are access structures on the physical level of the databases. They are indexes that allow the definition of a subset of tuples in a table through a conditional expression. This dissertation studies the identification and subsequent automatic creation of partial indexes that can contribute in improving the performance of a database system. We propose an algorithm that examines, for each relevant query, the indexable attributes set, for which the creation of a partial index could influence the query optimizer to generate plans that are more efficient. We perform data mining on indexable attributes patterns to obtain correlated attributes according to their frequency in queries within the particular workload. We obtain a proposal for a set of candidate partial indexes considering also a benefit heuristics. We may consider a self-tuning analysis of an index configuration with both complete and partial indexes. We have implemented techniques and algorithms proposed in this research into DBX, a framework that allows local and global self-tuning regarding relational databases.

Keywords

databases; self-Tuning; partial indexes

Sumário

Lista de figuras

Lista de tabelas

1
Introdução

As aplicações de bancos de dados têm se tornado cada vez mais complexas e variadas. Atualmente, estas aplicações podem ser caracterizadas pelo grande volume de dados acessados e pela elevada demanda por desempenho, no que diz respeito tanto à diminuição do tempo de resposta das consultas quanto ao aumento da vazão (número de consultas executadas por unidade de tempo). Neste contexto, a sintonia do projeto físico de bancos de dados tem se revelado extremamente importante na melhoria do desempenho dos sistemas de bancos de dados.

1.1.
Motivação

A sintonia fina de um banco de dados não se trata de uma tarefa trivial e pode envolver diversas estratégias relacionadas ao projeto físico, por exemplo, por meio da manutenção de índices, visões materializadas, particionamentos horizontais ou verticais de tabelas, replicação de dados, reescrita de consultas, entre outras. A sintonia de índices, como parte do projeto físico de banco de dados, consiste na tarefa de selecionar, criar, excluir e reconstruir estruturas de índices com o objetivo de reduzir o tempo de processamento das cargas de trabalho. Dentre as atividades relacionadas à sintonia de bancos de dados, o ajuste das estruturas de índices, sem dúvida, representa uma das mais relevantes. Este fato decorre do grande benefício que estas estruturas trazem para o desempenho dos sistemas de bancos de dados, pois pode reduzir substancialmente o tempo de execução das consultas, inclusive das atualizações [1].

Um tipo particular de índice são os índices parciais, que se encontram presentes em alguns dos principais Sistemas de

Gerenciamento de Banco de Dados (SGBD), tais como: PostgreSQL [2] e SQL Server [3], que afirmam que os índices parciais trazem melhorias no desempenho da execução de consultas em relação aos índices tradicionais ou completos, nos casos em que há exigências de informações focadas em subconjuntos dos dados ou por conta de existirem termos específicos de busca em certos conjuntos de dados.

A título de exemplo, se há uma tabela com as mensagens que os usuários enviam para outros usuários em um *chat* e na qual haja o atributo *mes_unread* com valor 1 e se a mensagem foi lida pelo destinatário e 0 no caso contrário, costuma-se pensar que a quantidade de mensagens não lida pelos usuários é menor que a lida. Portanto, é válido criar um índice parcial naquelas tuplas cujo valor do campo *mes_unread* é igual a 0. Poder-se-ia ter um índice com tamanho menor que um índice completo, que reduziria o espaço de busca no índice e, por conseguinte, evitaria a análise de tuplas não relevantes para uma consulta. Daí advém a ideia que os índices parciais podem ser uma opção desejável na sintonia fina do projeto físico de um banco de dados.

1.2. Objetivos e escopo da dissertação

O objetivo deste trabalho encontra-se no processo de sintonia fina para índices parciais. A abordagem objetiva a seleção de configurações de índices parciais com uma grande possibilidade de serem usados nos planos de execução das cargas de trabalho. Os objetivos específicos pretendidos nesta dissertação compreendem:

- Estudar as principais diferenças dos índices parciais e completos que tornam os índices parciais um importante recurso na hora de fazer sintonia fina em bancos de dados;
- Elaborar um processo de seleção de índices parciais, voltados para os conjuntos de dados frequentemente acessados;
- Propor uma estratégia que permita complementar as ações de sintonia fina orientadas à criação de índices parciais com ações de

sintonia fina que envolvam outras estruturas de acesso. Neste caso, a ênfase reside na seleção de configurações de índices parciais e completos ao mesmo tempo;

- Implementar as estratégias e soluções propostas, avaliando-as segundo medidas de desempenho apropriadas.

O escopo deste trabalho não visa procurar a melhor configuração de estruturas de acesso possível para uma carga de trabalho, mas fornecer configurações alternativas aos índices completos que permitam a diminuição dos tempos de execução e/ou aumentem a vazão das cargas de trabalho.

1.3. Estrutura da dissertação

O Capítulo 2 aborda as definições e conceitos necessários ao entendimento desta dissertação e do contexto em que se insere. Além disso, definem-se os índices parciais, que são possíveis estruturas de acesso a serem consideradas pelo otimizador na execução das consultas em uma carga de trabalho. No capítulo também se discute algumas estratégias de sintonia fina referentes aos índices parciais e completos.

No Capítulo 3 discorre-se sobre uma proposta de procedimento para realizar sintonia fina orientada a selecionar índices parciais potencialmente úteis na execução das consultas de uma carga de trabalho. Demonstra-se o modelo de custos considerado no trabalho para os índices parciais e a estratégia usada para considerar as configurações envolvendo tanto índices completos como parciais.

Enquanto no Capítulo 4 apresenta-se não só a ferramenta que será implementada a proposta deste trabalho, bem como o cenário onde são desenvolvidos os testes e a metodologia de avaliação com base nas medidas de desempenho.

2
Conceitos e fundamentos

Este capítulo descreve sucintamente os principais conceitos necessários para o entendimento do presente texto. Está organizado para abordar os principais assuntos envolvidos nesta tese, a saber, sintonia fina de banco de dados com índices parciais.

2.1.
Estruturas de acesso

O tempo de resposta na execução de consultas em um banco de dados encontra-se intimamente relacionado com o projeto físico do banco de dados, o que inclui o particionamento de tabelas, o pré-processamento, a duplicação de subconjuntos de dados e até os mecanismos utilizados para acelerar o acesso aos dados, os quais são chamados de estruturas de acesso [4]. O conjunto de estruturas de acesso disponíveis em um banco de dados, em um determinado instante de tempo, determina a própria configuração dessas estruturas de acesso [4].

Uma estrutura de acesso utilizada para fins de otimização é um índice que permite uma localização rápida de uma tupla ou conjunto de tuplas quando efetuada uma consulta. Os sistemas de gerenciamento de dados utilizam os índices de maneira semelhante ao índice remissivo de um livro onde se verifica um determinado assunto e, depois, localiza-se a sua posição em uma dada página. A utilidade dos índices numa determinada busca depende da organização estabelecida na chave do índice, o que contribui na busca de valores que satisfaçam a consulta. Por exemplo, a estrutura física de alguns índices é constituída por uma árvore B+-tree que cria uma correspondência entre os valores da chave do índice e as tuplas a que pertencem. Além disso, faz-se uma ordenação dos valores da chave do índice, permitindo encontrar rapidamente aqueles valores que satisfazem uma chave de busca.

Para acessar os registros de uma tabela através de índices, utilizam-se os métodos de acesso. Pode-se mencionar os métodos *index scan* e *bitmap index scan*, que servem para a busca de tuplas específicas ou conjuntos de tuplas, além da combinação dos resultados obtidos de múltiplos índices.

No caso da operação *index scan,* o SGBD procura em uma mesma faixa de valores segundo a chave do índice. Por exemplo, no caso de uma busca em um índice com estrutura física constituída por uma árvore B+-tree, é localizado no índice o menor ou o maior valor que satisfaz a chave de busca e, depois, procede-se uma busca dos outros valores da faixa na ordem em que se encontram no índice.

Por outro lado, a operação *bitmap index scan* procura os valores da chave do índice que satisfazem a chave de busca da mesma forma que *index scan* e cria um *bitmap* em que cada *bit* é associado a uma tupla ou a uma página da tabela (caso a tabela tenha muitas páginas) que satisfaz a chave de busca ou contém, no mínimo, uma tupla que satisfaz a chave de busca, respectivamente. A efetividade deste mecanismo é que podem ser combinados os resultados das operações de acesso e as tuplas podem ser acessadas na ordem em que se encontra na tabela. Assim, uma vez lida uma página, são processadas todas as tuplas.

Por exemplo, na Figura 2.1 apresenta-se uma tabela com as tuplas e os blocos aos quais pertencem as tuplas. Suponha um índice B+-tree com chave “tutor” em que a ordenação dos valores da chave faz-se em ordem alfabética e, portanto, a ordem em que se encontram indexadas as tuplas não seja a mesma da tabela. Se o índice acessa-se com a operação *bitmap index scan* e a chave tutor=’Ebo’, obtém-se o bitmap:

(1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, ...)

Caso se precise dos nomes das pessoas com tutor=’Ebo’, deve-se acessar as tuplas da tabela, mas, dado que se tem o *bitmap* obtido da operação *bitmap index scan* enquanto se carrega, na memória, os blocos do bitmap pode-se acessar todas as tuplas envolvidas no bloco que satisfazem a chave de busca permitindo que o bloco seja lido do disco uma vez só.

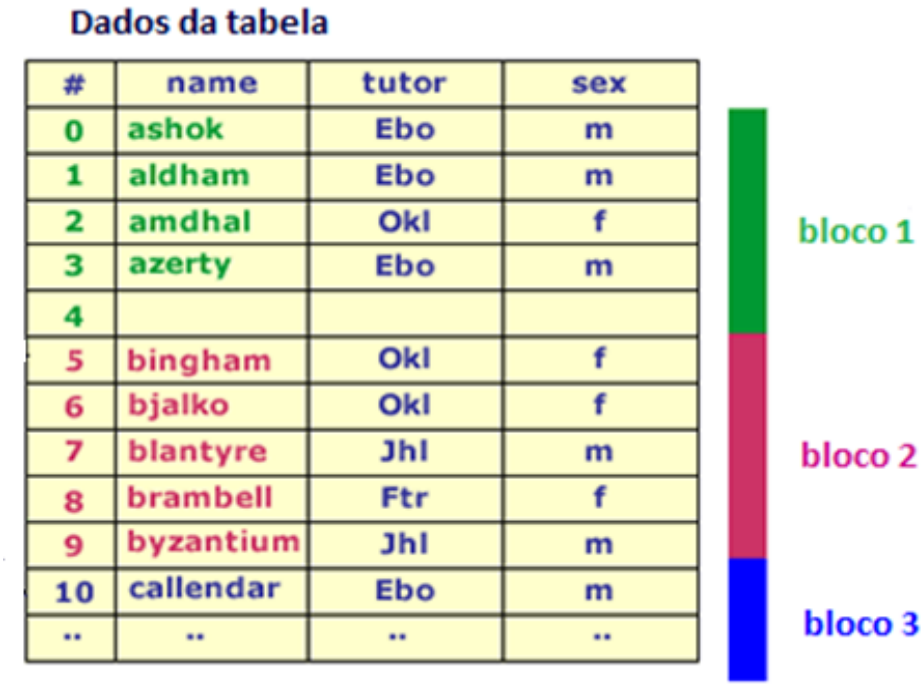


Figura 2-1: Tabela organizada em blocos [5]

2.2. Índices parciais

Os índices parciais foram mencionados pela primeira vez por Michael Stonebraker [6], que os define como índices construídos em um subconjunto das tuplas de uma tabela definido por uma expressão condicional. Por exemplo, no atributo *SalesAmount* de uma tabela *Sales*, pode-se definir um índice parcial chamado de *Sales_SalesAmount* como segue:

```
CREATE INDEX Sales_SalesAmount
ON Sales (SalesAmount)
WHERE SalesAmount < 10
```

Note-se que, neste caso, a sintaxe de construção de índices parciais é similar ao caso dos índices completos, com a exceção que se agrega a cláusula *WHERE* que define o subconjunto de tuplas a serem indexadas. A expressão condicional definida nos índices parciais pode reduzir a utilização destes, desse modo, um índice parcial P pode ser usado na execução de uma consulta Q se e só se o predicado de Q implique

logicamente o conjunto de expressões condicionais do índice P. Por exemplo, seja a seguinte consulta:

```
SELECT SalesAmount
FROM Sales
WHERE Date BETWEEN '01/01/2010' AND GETDATE() AND
SalesAmount BETWEEN 4 AND 8
```

O predicado da consulta expresso na cláusula *WHERE*, que restringe o atributo *SalesAmount* na faixa de valores 4 ao 8, contemplado na faixa de valores definida na expressão condicional do índice. Assim, o índice parcial é uma possível opção a ser considerada pelo otimizador para o acesso dos dados procurados pela consulta. Contudo, se a faixa de valores procurada na consulta fosse 4 a 12, então, o índice parcial não poderia ser usado. Portanto, a principal desvantagem dos índices parciais é que, se forem mal concebidos, podem existir consultas de alto custo de processamento que requerem tuplas não indexadas, causando varreduras completas nas tabelas, gerando um grande impacto sobre o desempenho da execução das consultas. Os índices parciais são úteis nos casos em que há requisitos de informação focados em subconjuntos dos dados de uma tabela ou há uma diferenciação entre os dados procurados com frequência e os dados que raramente são consultados. Exemplos das situações referidas encontram-se em empresas de serviços bancários que apresentam informações aos clientes sobre as transações mais recentes feitas ao longo de um período de tempo ou empresas de serviços em que se precisa da análise dos serviços não realizados. Nestes exemplos, os dados das ultimas transações bancárias resultam dados consultados frequentemente ou ativos e têm-se requisitos informacionais nos serviços não realizados.

A melhoria do desempenho na execução das consultas pela utilização dos índices parciais justifica-se pela redução da quantidade de leituras lógicas como uma consequência direta da redução do espaço de busca na estrutura de acesso. Por exemplo, no caso da tabela *orders* do *benchmark* TPC-H [7] com o esquema:

```
CREATE TABLE orders
( o_orderkey serial NOT NULL,
 o_custkey bigint NOT NULL,
 o_orderstatus character(1),
 o_totalprice numeric,
 o_orderdate date,
 o_orderpriority character(15), (E1)
 o_clerk character(15),
 o_shippriority integer,
 o_comment character varying(79),
 CONSTRAINT orders_pkey PRIMARY KEY (o_orderkey),
 CONSTRAINT orders_o_custkey_fkey FOREIGN KEY (o_custkey)
 REFERENCES customer (c_custkey) MATCH SIMPLE
 ON UPDATE NO ACTION ON DELETE NO ACTION)
```

No acesso aos dados através da consulta:

```
SELECT *
FROM orders
WHERE o_orderpriority = 'L' and (Q1)
 o_orderstatus = 'O'
```

Se usado um índice parcial com a chave *o_orderstatus* e a expressão condicional *o_orderpriority = 'L'*, com só encontrar no índice os registros de tuplas que satisfazem *o_orderstatus = 'O'* pode-se atender à consulta Q1. Contudo, se a estrutura de acesso usada for um índice completo com a chave *o_orderstatus*, a busca no índice iria encontrar os registros de tuplas que satisfazem *o_orderstatus = 'O'*, mas sem qualquer informação sobre a condição *o_orderpriority = 'L'*. Portanto, o número de registros de tuplas devolvidas e a quantidade de leituras lógicas seriam maiores, o que implicaria maior tempo de processamento.

Um exemplo prático da melhoria de desempenho possível de ser obtida com os índices parciais é o seguinte. Considere a tabela *orders* do *benchmark* TCP-H, com o esquema (E1) e 4 500 000 tuplas. Os valores possíveis do atributo *o_orderpriority* são *"L"* com uma frequência do 20,20% e *"O"* com frequência do 49,27%. Agora, caso se crie os seguintes índices parciais, um de cada vez:

```
CREATE INDEX partial_orderpriority
on orders (o_ orderpriority) (E2)
 WHERE o_ orderpriority = “OK”
```

```
CREATE INDEX complete_ orderpriority (E3)
on orders (o_orderpriority)
```

E se executa a seguinte consulta:

```
SELECT * FROM orders
WHERE o_orderpriority = 'LOW' and (Q2)
 o_orderstatus = 'OK'
```

Obtêm-se os planos de execução apresentados na Figura 2.2 para os índices *partial_orderstatus* e *complete_ orderpriority*, nessa ordem.

```
Bitmap Heap Scan on public.orders (cost=10375.95..99966.02 rows=447938 width=111)
 (actual time=916.400..8881.559 rows=438356 loops=1)
 Recheck Cond: ((orders.o_orderpriority = 'LOW '::bpchar) AND (orders.o_orderstatus = 'OK'::bpchar))
 Heap Blocks: exact=42188
 Buffers: shared hit=3 read=43868
 -> Bitmap Index Scan on partial_orderpriority (cost=0.00..10263.97 rows=447938 width=0)
 (actual time=889.900..889.900 rows=438356 loops=1)
 Index Cond: (orders.o_orderpriority = 'LOW'::bpchar)
 Buffers: shared hit=2 read=1681
"Planning time: 0.354 ms"
"Execution time: 843.868 ms"

Bitmap Heap Scan on public.orders (cost=20939.04..117284.29 rows=447938 width=111)
 (actual time=257.644..1895.481 rows=438356 loops=1)
 Recheck Cond: (orders.o_orderpriority = 'LOW'::bpchar)
 Rows Removed by Index Recheck: 2296317
 Filter: (orders.o_orderstatus = 'O'::bpchar)
 Rows Removed by Filter: 461258
 Heap Blocks: exact=29970 lossy=52738
 Buffers: shared read=86158
 -> Bitmap Index Scan on complete_ orderpriority
 (cost=0.00..20827.06 rows=909150 width=0)
 (actual time=248.313..248.313 rows=899614 loops=1)"
 Index Cond: (orders.o_orderpriority = 'LOW'::bpchar)
 Buffers: shared read=3450
Planning time: 2.972 ms
Execution time: 1915.441 ms
```

Figura 2-2: Plano de execução da consulta Q2 com os índices E2 e E3

Com um acesso *bitmap index scan* no índice parcial *partial_ orderstatus*, obtém-se 438.356 tuplas e do índice *complete_ orderpriority*, 899.614 tuplas. Isto confirma que índices parciais bem desenhados diminuem a quantidade de leituras lógicas, neste caso quase a metade.

Por outro lado, existem técnicas de particionamento e indexação de tabelas que, em alguns casos, efetuam a mesma função que os índices parciais. Por exemplo, o Oracle e o SQL Server permitem fazer particionamento de tabelas e, logo, indexar as partições individualmente. Além disso, o Oracle possibilita a construção de índices globais em todos, ou em um subconjunto de partições de uma tabela. Contudo, apesar da utilidade destas técnicas de particionamento e indexação de tabelas, só complementam os índices parciais e vice-versa [8][9].

2.3. Sintonia fina em bancos de dados

A sintonia fina compreende as atividades de ajuste de configurações e parâmetros de um SGBD para prover o otimizador do SGBD de um ambiente que permita escolher um plano de execução para cada consulta da carga de trabalho, isto é, o conjunto de consultas a serem executadas mais dados presentes no banco de dados.

Geralmente, as atividades de sintonia fina em bancos de dados são feitas pelos DBA (*Database Administrators*) que são profissionais especializados encarregados de monitorar as variáveis de controle do SGBD (*e.g.* uso de memória RAM, acessos ao disco, tempo de resposta) e determinar, segundo a carga de trabalho, comandos que proporcionem a melhoria de desempenho. Por exemplo, podem ser feitas modificações nos parâmetros de controle envolvendo a quantidade de memória, o nível de concorrência, ou mesmo a seleção da configuração de estruturas de acesso [10] [11] [1] [12].

Tendo em vista que encontrar uma configuração ótima do projeto físico de um banco de dados não corresponde a uma tarefa trivial [10], faz-se necessária a criação de ferramentas que automatizem as tarefas de sintonia fina. Estas ferramentas devem ser capazes de adaptar-se dinamicamente às modificações da carga de trabalho [13], [10].

A sintonia fina automática em bancos de dados refere-se ao ajuste automático de configurações buscando melhorar o desempenho por meio

da diminuição do tempo de resposta das transações ou comandos executados no SGBD [14].

A configuração da estrutura física dos bancos de dados, através da seleção de estruturas de acesso, reescrita de consultas, visões materializadas e particionamento de tabelas, tem recebido especial atenção nos últimos anos. Neste sentido foram realizados numerosos trabalhos de pesquisa [15], [16], [17], [18], DASI08, [19], [20], [14], nos quais se destacam três aspectos principais para compor ferramentas de sintonia fina no projeto físico de um banco de dados [10]:

- Espaço de busca: refere-se às possíveis estruturas de acesso candidatas a serem selecionadas por uma ferramenta automática para fazer parte da configuração física do banco de dados.
- Modelo de custo: é elaborado fora do otimizador do SGBD para realizar estimativas do custo de execução de comandos em configurações de estruturas de acesso candidatas ou reais.
- Estratégia de Enumeração: considera o espaço de busca de estruturas de acesso candidatas juntamente com o modelo de custo, a carga de trabalho e as limitações do ambiente para enumerar configurações de estruturas de acesso candidatas.

A seguir, apresentam-se alguns trabalhos e técnicas que podem ser usadas no processo de sintonia fina automatizada.

2.3.1. Indexação com índices parciais

A literatura a respeito dos índices parciais pode ser dividida em duas linhas temáticas. A primeira refere-se à seleção de estruturas de acesso [21] [22] [23] e a segunda, chamada de *database* Cracking [24], [25], [26], envolve o desenvolvimento de estruturas de acesso adaptáveis à carga de trabalho.

A primeira linha temática começou a ser desenvolvida por Seshadri e Swami (1995) [21], através de uma proposta na qual os índices parciais haviam sido criados em conjuntos de tuplas do banco de dados estatisticamente úteis no processamento da carga de trabalho. Os autores

se propõem, desse modo, a coletar diferentes níveis de informação, como atributos contidos nas consultas e estatísticas de uso destes, constantes envolvidas nas consultas e sua distribuição. Além disso, descrevem a estratégia de indexação utilizada que tenta dividir os recursos espaciais disponíveis para a criação de índices parciais segundo os seguintes passos.

1. Amostragem: cada tabela é mostrada para criar um histograma de valores para cada coluna de cada tabela. Esta técnica é usada para fazer estimativas da quantidade de tuplas envolvidas em faixas de valores nas colunas.
2. Determinar intervalos de exclusão e inclusão: os intervalos de exclusão são conjuntos de tuplas não contidas nos índices parciais e os intervalos de inclusão são aqueles que serão indexados. No processo seguido na obtenção destes intervalos, o domínio de cada coluna é dividido em faixas de igual tamanho. Um valor limiar T é escolhido para cada coluna e, em seguida, verifica-se para cada intervalo se a quantidade de tuplas contidas (o histograma é usado para estimar o número de tuplas contidas no intervalo) é maior que o valor T correspondente. No caso de ser maior, o intervalo é classificado de exclusão, T é modificado segundo um ou vários fatores de ponderação limiar (Threshold Weighting Factors) que fazem uso das informações de estatística, e recomeça-se o processo de classificação das faixas classificadas de exclusão até esgotarem-se os recursos disponíveis (espaço em disco e número de índices) para a criação de índices parciais.
3. Geração de índices: os índices são criados de acordo com os intervalos de inclusão e exclusão.

Outros trabalhos foram desenvolvidos no âmbito de adaptações das técnicas apresentadas anteriormente dependendo do cenário de aplicação. Por exemplo, a proposta de Seshadri e Swami (1995) [21] foi adaptada para ambientes distribuídos e dinâmicos baseados em Peer to Peer (P2P) [22].

A tecnologia P2P embora seja usada com sucesso em muitas aplicações para localizar o conteúdo, tem sido menos eficaz ao lidar com grandes quantidades de dados, por causa do elevado impacto na manutenção de índices que, às vezes, tem tamanho proporcional ao tamanho dos dados. Razão pela qual, a proposta apresentada por Wu e colaboradores [22], contrária ao sistema de indexação P2P tradicional que indexa todos os dados, identifica subconjuntos de dados a serem indexados com base em critérios como: frequência da consulta, frequência de atualização, custo de indexação etc.

Por outro lado, Lu e colaboradores [23] realizam a seleção automática de índices parciais em grandes quantidades de dados. Em seu artigo, os autores propõem a redução do tamanho de índices usando técnicas de compressão e, no caso em que estes índices compactados tiverem um alto custo de manutenção, é sugerida a criação de índices parciais *bitmap* nos conjuntos de dados frequentemente consultados baseados no algoritmo de indexação proposto por Seshadri e Swami (1995). A detecção dos conjuntos de dados frequentemente consultados é realizada com o uso de histogramas de consulta para cada atributo.

Apesar das grandes inovações aportadas nos trabalhos anteriormente apresentados, estes não consideram a correlação de uso dos atributos nas consultas e, geralmente, possuem independência estatística na hora de contabilizá-la nos atributos. Esta independência não representa cenários realistas, onde os usuários estão interessados em várias características dos dados e são elaborados relatórios destes. Entretanto, existe uma correlação de uso dos atributos das consultas. Por exemplo, as equipes de vendas de uma companhia podem ter interesse no estudo do ganho em relação ao custo das vendas, isto pode resultar na correlação destes atributos na carga de trabalho.

Por conseguinte, os trabalhos existentes na literatura desconsideram situações em que os índices parciais não são úteis, como mencionado na seção 2.2 e faz-se necessário um enfoque que permita misturar ações de sintonia fina com índices parciais e ações de sintonia fina com outras estruturas de acesso.

A segunda abordagem envolve a estratégia intrusiva em bancos de dados, chamada *database cracking* [24], baseia-se na hipótese de que a manutenção do índice deve ser um subproduto do processamento da consulta, não de atualizações. Cada consulta é interpretada não só como um pedido de um conjunto de resultados específicos, como também uma ativação ou marcação dos dados procurados, o que contribui para a fragmentação ou particionamento do banco de dados. Cada pedaço do banco de dados ou conjunto de dados de interesse é descrito por uma consulta e montado em um índice particionado composto por subíndices criados para atender as consultas, o que resulta na prática em um índice parcial. Os dados não-indexados permanecem assim até que uma consulta os solicite. O índice quebrado é construído de forma dinâmica enquanto as consultas são processadas adaptando-se às mudanças na carga de trabalho.

Uma implementação de *database cracking* encontra-se no trabalho desenvolvido por Idreos, Kersten e Manegold (2007) [25] que apresentam a primeira arquitetura completa deste enfoque aplicada no contexto do banco de dados orientado a coluna MonetDB. Uma desvantagem da estratégia *database cracking* é que, no índice fragmentado, as chaves são divididas em faixas disjuntas sem uma ordem que as agrupe, ao contrário dos índices habituais que estabelecem uma organização de todos os valores da chave. Uma estratégia orientada à resolução desta desvantagem é a *adaptive merging* [27] que combina a eficiência da criação de árvore tradicional B+-tree com o comportamento adaptativo e incremental de *database cracking*. A essência desta estratégia consiste em usar as capacidades das árvores B+-tree e o comportamento adaptativo e incremental de *database cracking* criando um índice B+-tree para cada pedaço do índice, uma vez que os dados sejam procurados nesses índices, combiná-los segundo a estratégia de ordenação.

O problema principal das estratégias apresentadas na segunda abordagem é que não se consideram os casos em que a criação de índices completos faz mais sentido para a carga de trabalho. Isso decorre da falta de uma análise da frequência de uso dos índices fragmentados /parciais. Portanto, pode-se ter índices que não trazem muitos benefícios

para a carga de trabalho, porém a construção gradual destes índices fragmentados/parciais pode ir até a criação de índices que indexam a maioria dos valores de uma chave e as varreduras graduais feitas nos dados não indexados pode incorrer em um maior custo de execução.

2.3.2. Deteção de padrões de atributos frequentes

Um elemento importante quando se estuda a carga de trabalho em bancos de dados é a análise da frequência de certos padrões nas consultas. Neste viés, existem trabalhos realizados que consideram a mineração de elementos frequentes (*mining frequent itemsets*) [28], [29]) e fornecem técnicas que podem ser aplicadas na análise da frequência dos atributos presentes em consultas de uma carga de trabalho. Alguns exemplos da utilização destas técnicas na sintonia fina encontram-se em trabalhos como os de Agrawal, Chaudhuri e Narasayya (2000); Aouiche, Darmont e Gruenwald (2003); Azefack, Aouiche e Darmont (2008); Nadimi-Shahraki, Shahriari e Jazi (2015), que propõem técnicas para enumerar índices e visões materializadas utilizando conjuntos de atributos consultados frequentemente.

A seguir, formalizam-se alguns termos utilizados na mineração de elementos frequentes segundo a notação apresentada em Gouda e Zaki (2001):

Seja $I = \{i_1, i_2, \ldots, i_m\}$ um conjunto de *m* elementos distintos, *D* um banco de dados transacional, em que cada transação tem um identificador único (*tid*) e é composta por um subconjunto de elementos de I, e T={t1, t2, ..., tn} o conjunto de todos os identificadores (*tids*).

Então se chama:

- *itemset* em I: a todo conjunto X tal que $X \subseteq I$.

- *tidset* de X: a função t(X) que associa cada itemset X com o subconjunto de T de todas as transações (*tids*) que contêm o subconjunto X.

- Suporte de um *itemset* X: a função σ(X) que calcula a quantidade de elementos em t(X) (σ(X)=|t(X)|).

- *itemset* frequente: aqueles itemset X que satisfazem σ(X) > min_sup, onde min_sup é um valor limite a partir do qual considera-se um itemset frequente.

A mineração de elementos frequentes nas consultas de uma carga de trabalho consitui uma ferramenta poderosa na hora da análise da carga de trabalho. O conjunto com outras estratégias ou heurísticas pode ajudar a encontrar estruturas de acesso que aumentem o *throughput* das consultas.

2.3.3. Heurística de benefícios

Uma estratégia usada no processo de sintonia fina automática compreende a heurística de benefícios [30], que propõe a criação de estruturas de acesso no banco de dados quando estas contribuem de forma positiva nas consultas executadas neste banco. A heurística de benefícios já foi utilizada em outras pesquisas de sintonia fina [31], [32], [13], [33], [14], [34]. A ideia principal desta heurística consiste em atribuir benefícios às estruturas de acesso inexistentes no banco de dados chamadas de hipotéticas, quando se estima que uma consulta otimizaria o desempenho da carga de trabalho, se a estrutura de acesso hipotético existisse fisicamente no banco de dados. A efetiva criação da estrutura de acesso é determinada quando o benefício acumulado de sua utilização é superior ao seu custo de criação no banco de dados.

A seguir apresentam-se alguns conceitos envolvidos na heurística de benefícios.

Estrutura Real ou Materializada: uma estrutura real (ou materializada) é aquela que existe fisicamente, ocupando espaço em disco e que pode ser utilizada efetivamente para o acesso aos dados.

Benefício: o benefício $B_{ej,qk}$ de uma estrutura (candidata ou não) e_j no processamento de uma tarefa q_k pode ser formalmente definido como:

$B_{ej,qk} = \max(0, cost(q_k) – cost(q_k, e_j))$

Onde $cost(q_k)$ é o custo de execução da tarefa q_k sem utilizar a estrutura e_j; e $cost(q_k, e_j)$ é o custo de execução da tarefa q_k utilizando a estrutura de acesso e_j.

Benefício acumulado: o benefício acumulado de uma determinada estrutura de acesso corresponde à soma de todos os benefícios para cada uma das tarefas anteriormente executadas.

2.4. Sintonia fina local e global

Atividades de sintonia fina (*tuning*) realizam-se segundo múltiplos parâmetros e gera múltiplas alternativas que podem ser estudadas em separado ou em conjunto. Os trabalhos desenvolvidos na literatura referentes à sintonia fina automática podem ser divididos em dois grandes grupos [13]: o primeiro grupo refere-se aos trabalhos de sintonia fina automática local, que abordam problemas específicos, em separado, tais como: índices parciais, estabelecimento de níveis de concorrência, políticas de substituição de páginas para um gerenciador de *buffer*, entre outros.

No segundo grupo encontram-se trabalhos que estudam o impacto no sistema como um todo das ações de sintonia fina. Este tipo de abordagem tenta encontrar um equilíbrio entre configurações locais. As duas classificações consideram um dos cinco princípios básicos de sintonia fina em banco de dados “Think globally; fix locally” [1], que recomenda soluções locais pensadas em um âmbito global.

Existem alguns trabalhos desenvolvidos na área das estratégias de ajuste automatizado [16] que propõem estratégias para lidar simultaneamente com índices e visões materializadas. Por outro lado, Bellatreche, Karlapalem e Schneider (2000) desenvolvem um trabalho no âmbito de sintonia fina global que considera a distribuição espacial de índices e visões materializadas. O algoritmo proposto começa com uma

solução inicial repensada iterativamente com o objetivo de reduzir custos de execução redistribuindo o espaço de armazenamento entre os índices e as visões materializadas.

Além disso, Kamel Aouiche e Jérôme Darmont (2009) tendo em vista algumas idéias básicas dos trabalhos anteriormente mencionados, desenvolveram um algoritmo que, a partir de uma configuração candidata Oc (conjunto de estruturas candidatas), selecionam uma configuração tal que reduz custos de execução da carga de trabalho. Para isto começam com uma configuração inicial ou vazia à qual se agrega iterativamente as estruturas candidatas que, combinadas com o conjunto inicial, alcançam o maior benefício.

A seguir, ilustra-se uma situação onde se evidenciam técnicas de sintonia fina local e global orientada à seleção de índices e visões materializadas.

Considere uma empresa de vendas pela internet com um banco de dados que contém uma tabela *lineitem* com milhões de tuplas e o atributo *l_linestatus* que contém informação do *status* das entregas de produtos (êxito ou fracasso). Os executivos do Departamento de Vendas da empresa especificaram o requerimento informacional de conhecer as perdas da empresa por conceito de falhas na entrega dos produtos. Muitas consultas realizadas à base de dados oriundas do Departamento de Vendas têm como objetivo a tabela *lineitem* e, o atributo *l_linestatus* e, por consequência, são necessários mecanismos para que os tempos de resposta das consultas sejam mínimos.

Neste caso, poder-se-iam aplicar estratégias de sintonia fina local no banco de dados como a criação de índices ou visões materializadas utilizando o atributo *l_linestatus*, ou implementar soluções de sintonia fina global considerando a melhor alternativa ou se for possível a combinação das soluções.

Do exemplo anterior conclui-se que, enquanto as estratégias de sintonia fina local consideram espaços de busca específicos, as estratégias de sintonia fina global devem levar em conta todas as configurações possíveis e, portanto, todo o espaço de busca.

2.5.
Conclusões

Neste capítulo foram apresentados conceitos, algoritmos e fundamentos necessários para contextualizar a pesquisa apresentada. Enfatizou-se o interesse nas estruturas de acesso índices parciais e alguns casos em que são benéficos. Além de abordar técnicas e trabalhos de sintonia fina no caso dos índices parciais.

Acrescente-se que foram identificados elementos que fazem dos índices parciais um recurso atrativo para a sintonia fina e algumas das principais desvantagens dos trabalhos de sintonia fina. Nos próximos capítulos discorre-se sobre uma metodologia para se encontrar índices parciais relevantes na execução de uma determinada carga de trabalho.

3
Índices Parciais e Sintonia Fina

Devido à dificuldade de selecionar estruturas de acesso no processo de sintonia fina, foram desenvolvidas investigações no âmbito de técnicas e ferramentas que ajudam a fazer o ajuste automático ou semiautomático em bancos de dados. Algumas destas pesquisas foram desenvolvidas em matéria da sintonia fina com índices parciais como mostrado na seção 2.3.1.

Neste capítulo, propõe-se uma metodologia que, a partir da análise da carga de trabalho, encontra índices parciais que ofereçam benefícios na execução de uma determinada carga de trabalho. A metodologia volta-se para os conjuntos de tuplas acessados frequentemente na carga de trabalho para os que a criação de índices parciais diminua o tempo de processamento das consultas.

3.1.
Procedimento para realizar sintonia fina com índices parciais

Nas seções a seguir será abordada a proposta deste trabalho para efetuar sintonia fina utilizando índices parciais com base nos principais componentes identificados nas ferramentas de sintonia fina orientada à seleção de estruturas de acesso.

3.1.1.
Espaço de busca

Na seção 4.1 será mencionado que os índices parciais são úteis para indexar subconjuntos de dados frequentemente consultados. Assim, o espaço de busca dos índices parciais deve ser reduzido nesses subconjuntos. Além disso, mencionou-se no Capítulo 2 que a detecção de padrões frequentes nas consultas de uma carga de trabalho pode ocorrer

por meio da associação entre atributos das tabelas e conjuntos de itens como mencionado na seção 2.3.2. A solução para se achar o espaço de busca dos índices parciais deve-se aos seguintes passos, detalhados paulatinamente no resto da seção.

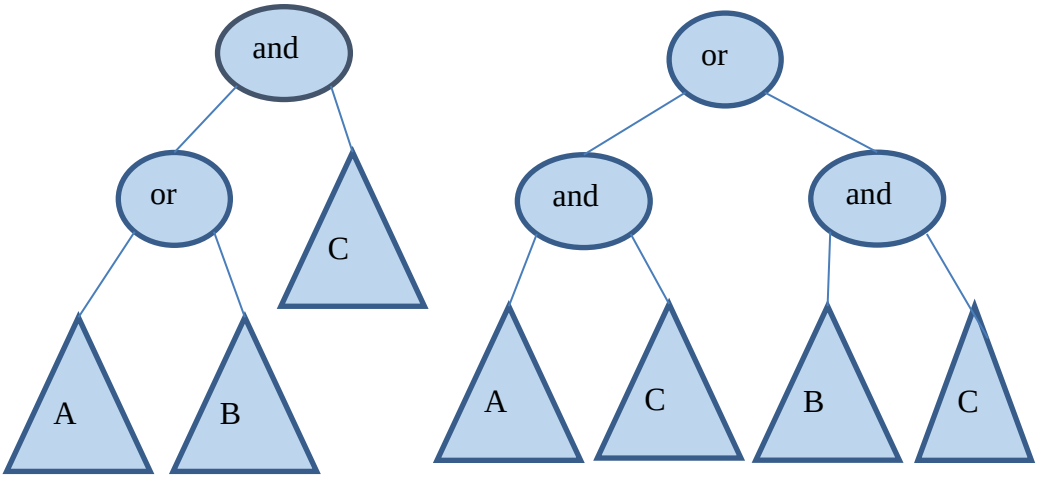
1. Para cada consulta da carga de trabalho extrair atributos indexáveis e atualizar o banco de dados transacional;
2. Encontrar conjuntos de atributos consultados frequentemente e propor índices parciais candidatos.

3.1.1.1. Extração de Atributos Indexáveis

No processamento das consultas, os atributos indexáveis são aqueles para os quais faz sentido a criação de índices. Assim, esta abordagem realiza uma análise dos predicados das cláusulas WHERE das consultas obtendo cláusulas equivalentes destas expressas como disjunção de conjunções e, para cada disjunção, gerar conjuntos de atributos indexáveis pertencentes à mesma tabela.

Dado um predicado P_1 é possível obter um predicado equivalente P_2 expresso como disjunção de conjunções se se representar o predicado P_1 como uma árvore binária com nós internos representando as operações *and*, *or* e *not*, restrições nas folhas e realizam-se transformações na árvore aplicando, sucessivamente, a regra lógica:

(A or B) and C ⇔ *(A and C) or (B and C)* no predicado da consulta, o que equivale a executar uma busca DFS na árvore binária procurando nós interno tipo *and* com nó filho tipo *or* (Figura 3.1a) e aplicar a transformação representada na Figura 3.1 (b) até obter uma árvore na qual os nós tipo *and* só têm filhos tipo *and*.



(a) (b)

Figura 3-1: Rotação da árvore segundo operações lógicas

Além disso, os conjuntos de atributos indexáveis são obtidos procurando os conjuntos de restrições com atributos em uma mesma tabela que pertencem a uma subárvore com nó raiz tipo *and* que na árvore binária modificada tem nó pai tipo *or*. Por exemplo, seja o predicado de uma consulta Q expresso como um conjunto de disjunções da seguinte forma:

p = (A = m and B = n) or (C = x and D = y) or (E = w and F =z) com todos os atributos pertencentes a mesma tabela. Então é possível extrair três conjuntos de atributos indexáveis de Q que representam três transações como mostrado na Tabela 3.1.

Tabela 3-1: Conjuntos de atributos indexáveis da consulta Q

Id de transação	Consulta	Atributos indexáveis
1	Q	AB
2	Q	CD
3	Q	EF

A heurística de extração de conjuntos de atributos indexáveis baseia-se na geração de conjuntos de atributos, a partir dos quais se

pode construir índices úteis para a indexação das tuplas referenciadas pelas cláusulas usadas para gerar atributos indexáveis. Por exemplo, no caso ilustrado, da primeira cláusula foi gerado o conjunto de atributos indexáveis {AB}, logo, os índices gerados com esses atributos podem ser utilizados para acessar os dados referidos nas cláusulas. Note-se que, no caso de cláusulas com operação *or*, não são gerados índices compostos baseado no fato de que o otimizador não faria uso destes [35].

3.1.1.2. Busca de Índices Parciais Candidatos

O processo proposto neste trabalho para encontrar subconjuntos dos dados de uma tabela focado nos requisitos de informações e, portanto, favoráveis à criação de índices parciais começa com a detecção de padrões frequentes nas consultas da carga de trabalho. A detecção de padrões de atributos frequentes já foi usada por Azefack e colaboradores [36] com o intento de achar conjuntos de atributos frequentes máximos (conjuntos de atributos frequentes que não são subconjunto de nenhum) na carga de trabalho e propor índices candidatos segundo o espaço disponível em disco rígido para a criação de índices. A proposta dos autores faz uso do algoritmo incremental GemMax [37] que faz uma busca *backtrack* nas novas transações adicionadas ao banco de dados transacional procurando itemset frequentes através de algumas otimizações feitas na busca.

Alguns algoritmos orientados à mineração de padrões frequentes utilizam o conceito de *conceptual lattices* [38], ou seja, conjunto de conceitos com uma ordem parcial (no caso específico, a relação de subconjunto define a ordem parcial nos itemset), que pode representar-se através de um grafo conceitual, onde os nós são constituídos pelos itemsets e as arestas denotam a relação de subconjunto entre itemsets.

Na Figura 3.2, mostra-se um grafo conceitual formado a partir do conjunto de elementos I = {ABCD} e o banco de dados transacional

mostrado na Tabela 3.2 conformado pelos identificadores de transações D = {1234}. O grafo foi projetado com suporte maior ou igual a 3.

Tabela 3-2: Exemplo de banco de dados transacional

TID	Elementos
1	A B C
2	B C D
3	A B C D
4	A C D

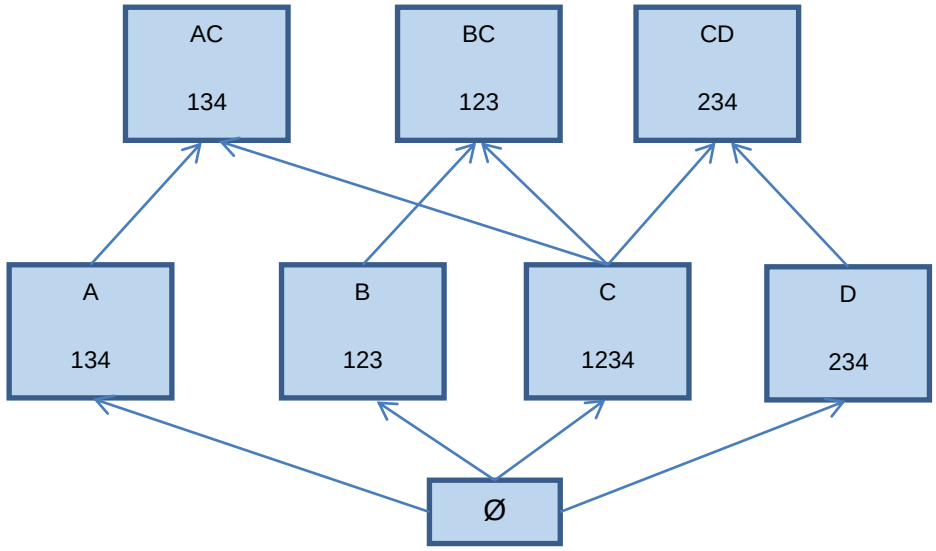


Figura 3-2: Exemplo de grafo conceitual

A adição de um itemset no grafo conceitual implica que o conjunto de atributos do itemset é frequente na carga de trabalho e, portanto, o conjunto de atributos e as restrições subjacentes das consultas da carga de trabalho poderiam servir como base na formação de um índice parcial. Daí a heurística para propor índices parciais candidatos baseia-se na adição de nós no grafo conceitual.

O processo proposto neste trabalho para obter padrões de atributos frequentes nas consultas de uma carga de trabalho começa por estabelecer uma correspondência entre os conjuntos de atributos indexáveis e as transações. Isto pode ser feito definindo uma função que forneça para cada atributo das tabelas de um banco de dados um valor

natural único e, assim, é possível representar os conjuntos de atributos indexáveis através de um conjunto de valores naturais ordenados que constituem as transações.

Seja A = {a_1, a_2,…, a_n} o conjunto de todos os atributos de um banco de dados e I = {i_1, i_2,…, i_n} um conjunto de elementos (itens, representado através de números naturais). Chama-se função de transformação de atributos indexáveis a função bijetora *f* que relaciona o atributo a_i em A com o elemento i_i em I.

Portanto, dado um conjunto de atributos indexáveis E={a_1,…,a_k}, a transação correspondente ao conjunto E será formada por uma ordenação do conjunto de elementos L = {f(a_1),…, f(a_k)}.

Por outro lado, a representação do banco de dados transacional é realizada utilizando a técnica de bitmap vertical [39], que associa cada *bit* de um *bitmap* com uma transação e, desse modo, se o *bit* for 1 então o elemento pertence à transação e se for 0 não pertence.

Em seguida, chama-se *bitmap* vertical ou representação de um banco de dados transacional ao conjunto de *bitmaps* B = {b_1, b_2,…, b_n} e a função bijetora *bitmapOf* que associa cada elemento do conjunto B com um item. Desse modo, dado um itemset I = {a_1, a_2, ... , a_k} é possível encontrar transações que contêm o itemset em base do *bitmap*:

$bitmapOf(\ a_1)$ & $bitmapOf(a_2)$ &…& $bitmapOf(a_k)$.

Além disso, se em um grafo conceitual de itemset frequentes houver uma aresta de um nó representante do itemset A para o nó representante de um itemset B, então A ⊆ B e, portanto tidset(A) ⊆ tidset(B), e por enquanto, B só pode ser frequente se A for frequente, daí que, em base na ordenação dos elementos de A e B possa se usar uma árvore com nós ordenados [40][41] para a representação do grafo conceitual. Uma abordagem semelhante a esta é utilizada por Valtchev e colaboradores [38].

A seguir, um exemplo da representação do grafo conceitual através de uma árvore com nós ordenados. Seja I = {A, B, C, D} um conjunto ordenado de elementos e Itemsets = {A, AB, AC, CD, ABC} um conjunto de itemsets frequentes. Então se pode obter a representação do grafo conceitual envolvendo os itemset em Itemsets através da árvore com nós

ordenados representada na Figura 3.3. Note que, na figura, representa-se nós terminais no fim dos itemsets e os nós encontram-se ordenados conforme I.

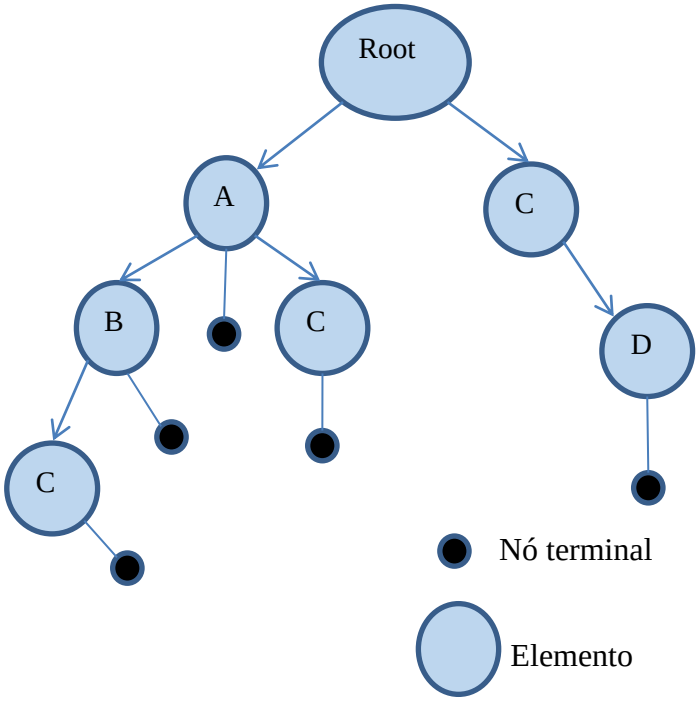


Figura 3-3: Exemplo de árvore com nós ordenados

Uma modificação feita na árvore de nós ordenados consiste em substituir os nós terminais por uma nova classe de nós que será denominada de virtuais. Os nós virtuais vão representar os nós que ainda não foram adicionados à árvore e terão uma variável com a quantidade de vezes que o nó se encontra em alguma transação. A ideia da modificação é levar em conta a frequência do itemset correspondente ao nó virtual no banco de dados transacional. Na Figura 3.4, mostra-se a árvore com nós ordenados modificada em correspondência com a árvore da Figura 3.3.

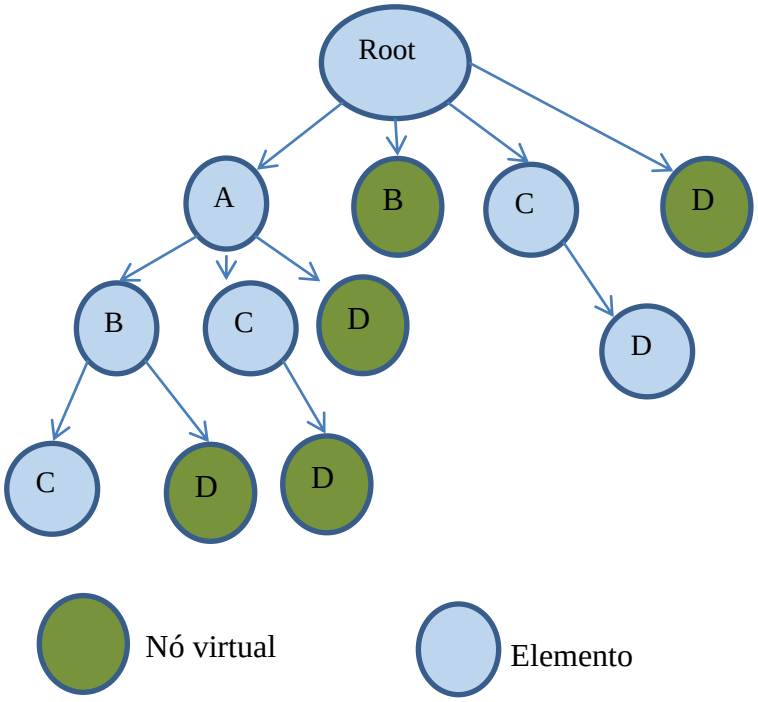


Figura 3-4: Exemplo de árvore com nós ordenados com a modificação

A obtenção dos itemset frequentes realiza-se através do uso da representação do banco de dados transacional e a representação do grafo conceitual. Neste processo, o primeiro passo consiste sempre em que se obtém uma nova transação do processo de transformação de atributos indexáveis em transações, adicioná-la ao banco de dados transacional, inserir no grafo conceitual os distintos padrões que podem ser obtidos da transação e acrescentar o contador de frequência dos nós virtuais. Assim, uma vez encontrado um nó virtual com contador de frequência maior que um valor limiar, o nó virtual passa ser o nó representante de itemset frequente e calculam-se os nós virtuais associados ao novo nó.

Na Figura 3.5 apresenta-se o algoritmo utilizado no processo de adição de uma nova transação no banco de dados transacional que recebe uma representação deste banco $B=\{b_1, b_2,\ldots, b_n\}$ e uma nova transação $t=\{a_1,\ldots, a_m\}$, verifica se a nova transação foi inserida no banco de dados transacional e insere um novo *bit* nos *bitmaps* se não for encontrada.

```
database_transform (B, t):
 Bitmap map = bitmapOfOne(B)
 foreach e in t
 map = map & bitmapOf(e)
 if not(haveAnyBitOne(map)) // Existe a transação em B?
 foreach b in B // Adicionar o bit 0 no final de cada
 // bitmap
 add_end_bit(b, 0)
 foreach e in t // Substituir o último bit por 1
 replace_last_bit(bitmapOf(e), 1)
```

Figura 3-5: Adição de novas transações no banco de dados transacional

Por outro lado, na Figura 3.6 apresentam-se dois algoritmos, a entrada do primeiro é uma árvore de nós ordenados representada pelo nó raiz A, uma transação T, uma variável com o índice do elemento sendo analisado na transação I, uma lista de itemsets frequentes L e a representação de um banco de dados transacional B = {b_1,…, b_n} faz um recorrido DFS (do inglês, *depth-first search*) na árvore. Isto segundo a ordem dos elementos na transação T encontrando nós virtuais que representam itemset que se tornaram frequentes e os transforma em regulares e gera os nós virtuais desses nós.

```
process_trie(A, T, I, L, B)
 if A is VIRTUAL
 A.freq++
 if A.freq > UMBBRAL
 L.add(build_itemset(A))
 A.transformToRegularNode()
 A.genNewVirtualNodes(B) // Gera novos nós virtuais
 // associados ao A
 else
 for(j = I+1; j < T.size; j++)
 if A.children.contains(T[j])
 process_trie(A.children.get(T[j]), T, j, L)

build_itemset(A)
 Itemset : Set
 while A != null
 Itemset.add(A.item)
 A = A.parent
 return Itemset
```

Figura 3-6: Processamento da árvore ordenada

Finalmente, o algoritmo representado na Figura 3.7 recebe como parâmetro uma transação e retorna um conjunto de índices parciais candidatos.

```
gen_Pindexes(t)
 indexes: Set
 itemsets: Set
 database_transform (B, t)
 process_trie(A, t, 0, itemsets)
 foreach itemset in itemsets
 indexes = indexes ∪ gen_partial_candidate(itemset)
```

Figura 3-7: Geração de índices parciais

Por outro lado, propõe-se fazer um *clustering* nas faixas de dados consultadas nos atributos contínuos. Este algoritmo aqui se baseia na proposta de [42], onde se propõe um método de *clustering* em faixas de dados segundo a função de similitude:

$D(x_i, x_j) = (a_i - c_j)^2+(b_i - d_j)^2$, se $x_i = <a_i, b_i>$ e $x_j = <c_j, d_j>$

O algoritmo implementado neste trabalho é incremental e faz uso da função de similitude para determinar a faixa mais próxima à nova faixa. Feito isso, mistura as duas faixas em uma só se a proporção de tuplas contidas na faixa misturada for menor que 20%. Caso essa mistura não seja realizada, procura-se a faixa seguinte mais próxima e o processo é repetido até misturar a nova faixa ou acrescentá-la ao conjunto de faixas.

O processo para se obter a expressão condicional do índice parcial consiste em atribuir para cada item uma lista ordenada conforme a ordem das transações com as restrições do atributo correspondente ao item. Assim, para conhecer a restrição correspondente ao item I em uma transação T deve-se apenas procurar o número de transações prévias a T com o item I e indexar na lista associada ao I. Desta forma, obtêm-se conjunto de conjunções associadas ao teste e às distintas transações permitindo expressar a condição condicional do índice parcial substituindo as restrições dos atributos contínuos pela restrição resultante da faixa correspondente à agrupação de faixas consultadas do atributo. A seguir, apresenta-se um exemplo do processo para se obter índices parciais candidatos.

Seja D = {t_1, t2, ..., t_n} um banco de dados transacional, Q = {q_1, q_2,…, q_n} consultas associadas ao banco de dados transacional de modo que a transação t_i corresponde à consulta q_i, L um grafo conceitual obtido das transações em D, f uma função que associada ao elemento ti ∈ D com a correspondente consulta q_i ∈ Q e, finalmente, a função g que dado um itemset X e uma consulta q com predicado expresso como disjunção de conjunções, retorna a conjunção correspondente a X no predicado de q.

Então se um novo itemset X = {x_1, ..., x_m} for adicionado em L, os índices parciais candidatos são gerados com uma expressão condicional obtida pela disjunção dos elementos g (x, f (t_i)) com t_i ∈ tidset(X). Suponha o exemplo a seguir formado por uma carga de trabalho W com as seguintes consultas:

```
select A, B, C from t
 where A > x1 and B > y1 and C > z1 (q1)
select B, C, D from t
 where B > y2 and C > z2 and D > w2 (q2)
select A, B, C, D from t
 where A > x3 and B > y3 and C > z3 and D > w3 (q3)
select A, C, D from t
 where A > x4 and C > z4 and D > w4 (q4)
```

Na Tabela 4.1 mostra-se o banco de dados transacional D resultado da transformação de W e na Figura 4.3 o grafo conceitual associado ao D.

Agora, se uma nova consulta for adicionada à carga de trabalho, por exemplo:

```
select A, B, D from t where A > x5 and B > y5 and D > w5 (q5)
```

com os atributos indexáveis {ABC}, os itemset {AD} e {BD} tornaram-se frequentes e, portanto, segundo a heurística propõe-se índices parciais candidatos com as expressões condicionais seguintes:

(A > x_3 and D > w_3) or (A > x_4 and D > w_4) or (A > x_5 and D > w_5)(I1)

(B > y_2 and D > w_2) or (B > y_3 and D > w_3) or (B > y_5 and D > w_5)(I2)

Note-se que q_3, q_4 e q_5 implica logicamente I1 e q_2, q_3; e q_5 implica logicamente I2, de forma que, os índices parciais I1 e I2 poderiam ser usados pelas respectivas consultas.

Como os índices parciais têm duas partes fundamentais, a expressão condicional e a chave com que o índice é criado neste trabalho encontram sequências de atributos frequentes, pode-se considerar a chave do índice formada por todos os atributos incluídos na expressão condicional baseada nessa metodologia.

3.1.2. Estratégia de enumeração

Determinado o espaço de busca dos índices parciais em função da detecção dos conjuntos de tuplas consultadas frequentemente precisa-se de uma estratégia capaz de avaliar o impacto dos índices parciais nas consultas frequentes ou raras. Daí se adotar heurísticas de benefícios em ordem de atribuir benefícios nas estruturas candidatas e encontrar aquelas com uma contribuição positiva na melhora do desempenho na execução dos comandos.

O procedimento seguido pela heurística consiste em encontrar, para cada consulta da carga de trabalho, o conjunto de índices parciais candidatos proposto, no espaço de busca, que traga maior benefício para a consulta conforme a estimativa do modelo de custo. Além de atribuir para esses índices um benefício igual ao custo de execução da consulta sem o índice menos o custo de executar a consulta com o índice. Em detalhe, para cada consulta, o algoritmo obtém o predicado da cláusula WHERE em forma de disjunção de conjunções; e para cada conjunção do predicado encontra o índice parcial candidato que supõe maior benefício e atribui benefício para o índice. Assim, em cada consulta pode se atribuir um benefício para vários índices parciais candidatos.

O algoritmo mencionado precisa de mecanismos que propiciem os seguintes elementos:

1. Obter o predicado de cada consulta da carga de trabalho expresso como uma disjunção de conjunções;
2. Para cada conjunção no predicado de uma consulta encontrar o índice parcial candidato que supõe maior benefício como estrutura de acesso.

O primeiro elemento mencionado é possível por intermédio das técnicas propostas de rotação da arvore obtida para a expressão no predicado da cláusula WHERE da consulta como explicado na seção 3.1.1.

No caso do segundo elemento, para cada disjunção de restrições do predicado da consulta, encontra-se o índice parcial candidato que ofereça o maior benefício para o acesso dos dados consultados para cada conjunção segundo o modelo de custos. Este procedimento é viável devido aos mecanismos propostos no trabalho para levar qualquer predicado para uma disjunção de conjunções.

Considere-se que um índice parcial candidato pode se materializar se o benefício alcançado pelo índice for maior que o custo de criação. Logo, o processo de numeração de índices parciais candidatos irá acumular benefícios nas estruturas candidatas sob consultas da carga de trabalho. Se o benefício acumulado nas estruturas exceder o custo de criação da estrutura, então, esta é proposta para criação.

O algoritmo proposto é baseado em Kamel Aouiche e Jérôme Darmont [43] para encontrar configurações de visões materializadas e índices. Tem como entrada o conjunto de índices parciais candidatos I_p, o conjunto de índices completos I_c e o conjunto de consultas significativas W. Define as variáveis locais:

C={}, *betterCand*={} e *betterBen*=0

A partir dos seguintes passos:

1. Para cada estrutura candidata *c* tal que satisfaz $c \in Ip \cup Ic$ e $c \notin C$;
2. Calcular o benefício *b* da configuração $c \cup C$ na carga de trabalho *W*;
3. Se *b* for maior que *betterBen* então *betterBen* = *b* e *betterCand* = *c*;
4. Se a variável não muda, então, o algoritmo acaba com a configuração C e, no caso contrário, atribui C = C U *betterCand* e começa no passo 2;

5. Criar as estruturas candidatas com custo de benefício acumulado maior que o custo de criação.

A maior vantagem do algoritmo consiste em avaliar os benefícios dos índices parciais e completos isoladamente em cada passo, decidindo a nova estrutura candidata para ser adicionada na configuração candidata em base na carga de trabalho. Isto permite que dado um índice parcial e o correspondente índice completo calcule-se o benefício que traz o índice parcial e o índice completo na carga de trabalho considerando a configuração candidata até o momento para decidir quem oferece melhor benefício para carga de trabalho.

Uma otimização para o algoritmo proposto é manter para cada estrutura candidata da carga de trabalho a lista das consultas em que a estrutura traz benefício, assim, cada vez que for testada a adição de uma estrutura candidata em uma configuração candidata basta comparar se existe uma consulta para a qual a nova estrutura candidata aporta maior benefício que a estrutura candidata correspondente na configuração candidata.

Um exemplo do funcionamento do algoritmo é o seguinte:

Considere o conjunto de índices parciais candidatos $\{p_1, p_2, p_3\}$ e o conjunto de índices completos $\{c_1, c_2, c_3\}$, com c_i o índice completo correspondente a p_i. Considere também o conjunto de consultas $W = \{s_1, s_2, s_3\}$ no qual pode se atribuir para cada estrutura candidata pares ordenados consulta-benefício:

$p_1 \rightarrow \{<s_1,5>\}$, $p_2 \rightarrow \{<s_2,4>\}$, $p_3 \rightarrow \{<s_3,4>\}$, $c_1 \rightarrow \{<s_1,4>, <s_2, 2>\}$, $c_2 \rightarrow \{<s_2,3>\}$ e $c_3 \rightarrow \{<s_3,3>\}$

Para o espaço de busca o algoritmo, sugere a configuração {c1, p3} com benefício acumulado 10, que se trata de uma configuração em que cada estrutura de acesso tem benefício acumulado máximo. Outro enfoque para a seleção da configuração final consideraria obter o benefício máximo para cada consulta, por exemplo {p1, p2, p3} e, desse modo, seria obtida uma configuração com maior benefício que a proposta. Contudo, nessa configuração poderia existir um grande número de índices em que o custo de criação seja maior que o benefício.

3.1.3.
Arquitetura da solução

Depois de apresentados as componentes do procedimento de seleção de configurações de índices parciais e os algoritmos a serem implementados será definida como é feita a integração destas componentes. Para isso, primeiramente são listadas as diferentes fases do procedimento:

- Extração de atributos indexáveis;
- Seleção de índices parciais candidatos;
- Redução de condições em atributos contínuos;
- Atribuição de benefícios nos índices candidatos;
- Seleção de índices parciais e completos.

Observe-se que as primeiras quatro fases são usadas para encontrar o espaço de busca de índices parciais e a última fase corresponde ao processo de enumeração de índices parciais e completos. A integração destas fases é demonstrada na Figura 3.8 que exibe como primeiro passo a obtenção das consultas do banco de dados. Mais tarde, estas consultas são processadas pela fase de *Extração de atributos indexáveis* e são elaborados os conjuntos de atributos indexáveis usados na fase de *Seleção de índices parciais candidatos*. Em seguida, executa-se a *Redução de condições em atributos contínuos* e são obtidas novas faixas para os atributos contínuos, conforme o *cluster* a que pertencem e se propõem os índices candidatos. Logo, os índices candidatos recebem benefícios na fase de *Atribuição de benefícios nos índices candidatos*, se algum índice candidato obtém benefício maior que o custo de criação, então o índice parcial candidato é submetido para a fase de *Seleção de índices parciais e completos* para ser considerado para materialização física.

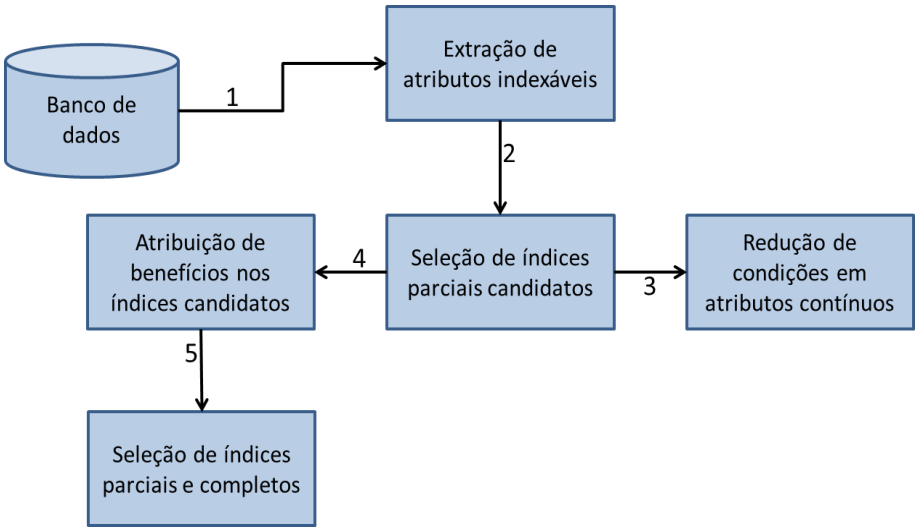


Figura 3-8: Arquitetura da solução proposta

3.2. Conclusões

Índices parciais bem projetados representam um recurso que pode resultar em uma grande melhora na execução da carga de trabalho em bancos de dados, por isso, neste capítulo se propõe um procedimento para encontrar configurações de índices parciais e completos com benefícios para uma carga de trabalho. O procedimento baseia-se na criação configurações de índices parciais e completos que permitam atender tanto as consultas mais frequentes como as menos frequentes de uma carga de trabalho.

O procedimento proposto neste trabalho é determinar o espaço de busca dos índices parciais considerando a correlação dos atributos nas consultas da carga de trabalho e uma heurística de benefícios, para a enumeração da solução usar uma estratégia que encontra uma configuração de índices parciais e completos a fim de atender a carga de trabalho.

No próximo capítulo irá se apresentar alguns testes desenvolvidos no procedimento proposto.

4
Implementação e resultados experimentais

Neste capítulo, apresenta-se a ferramenta de sintonia fina em que foi acoplado o procedimento proposto no Capítulo 3, o cenário dos testes realizados na ferramenta já modificada, os resultados obtidos nos testes e a avaliação dos resultados.

4.1.
Implementação do procedimento

Atualmente, existe uma grande quantidade de trabalhos e ferramentas na área de seleção automática do projeto físico de bancos de dados. Tais ferramentas, contudo, adotam uma abordagem *offline* na solução do problema e transferem para o DBA, dentre outras tarefas, a decisão de executar as recomendações sugeridas. Alguns destes trabalhos adotam uma abordagem intrusiva e funcionam apenas com um SGBD específico.

O procedimento proposto neste trabalho para a seleção de índices parciais não tem restrições enquanto à arquitetura da ferramenta em que vai se implementar, tanto faz se a abordagem for intrusiva ou não. Por isso, o critério de escolha da ferramenta para desenvolver o procedimento proposto baseia-se na extensibilidade da ferramenta de ajuste fino escolhida. Portanto, opta-se por uma ferramenta de sintonia fina atraente para a implementação da solução seja o DBX, que propõe uma abordagem não-intrusiva para a manutenção automática e *on-the-fly* do projeto físico de bancos de dados. A abordagem proposta é completamente desacoplada do código do SGBD, pela qual pode ser utilizada com qualquer SGBD e executada sem a intervenção humana.

O DBX segue o ciclo clássico de autossintonia: Observação, Predição e Reação como descrito inicialmente por Weikum e colaboradores [12]. Estas fases são continuamente executadas com a finalidade de: monitorar o comportamento corrente do sistema; derivar

decisões sobre o comportamento futuro; e, caso mudanças sejam necessárias, aplicá-las alterando as propriedades e o comportamento do sistema.

De modo geral, a arquitetura do DBX mostrada na Figura 4.1 se propõe a realizar a autossintonia durante a operação normal do SGBD, por meio da colaboração de agentes de *software*. O fluxo de execução na arquitetura do DBX começa com a captura da carga de trabalho através do agente Workload Obtainment que aloca informações da carga de trabalho em um repositório de meta-dados, as quais são usadas depois pelo agente encarregado de executar as heurísticas integradas para a manutenção das estruturas.

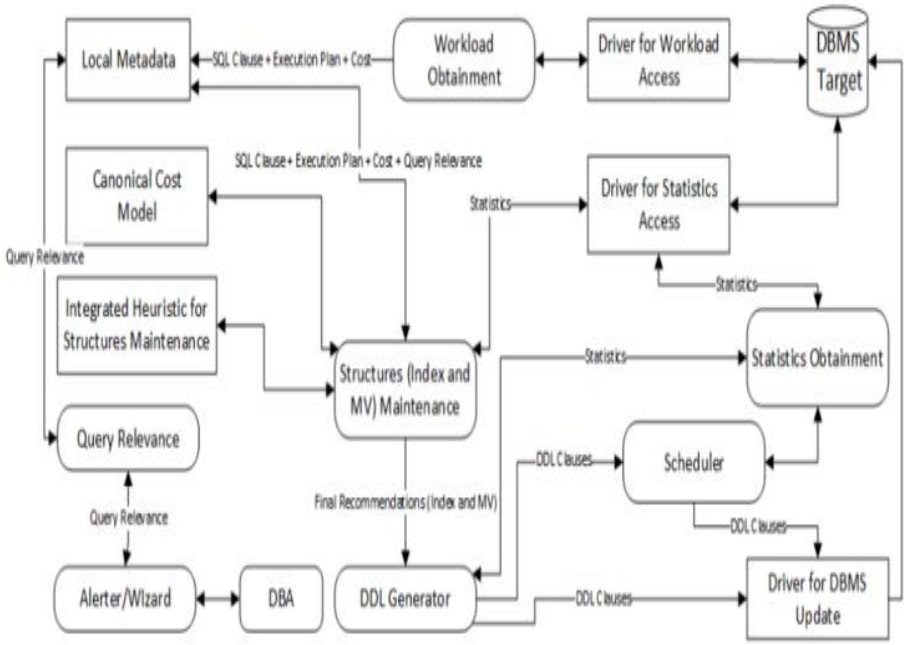


Figura 4-1: Arquitetura do DBX

Atualmente, o DBX tem implementado heurísticas para a criação e a manutenção de índices completos e visões materializadas isoladamente, ou seja, desconsidera que a carga de trabalho poderia se beneficiar de ações de sintonia fina que levem em conta combinadamente índices completos e visões materializadas.

A arquitetura do DBX pode ser facilmente estendida apenas acrescentando novas heurísticas e encontra-se desacoplada do código-fonte do SGBD permitindo a elaboração de planos de execução

hipotéticos, cujo resultado é que o DBX seja uma ferramenta a ser usada em diferentes SGBD.

Neste trabalho o DBX foi estendido com duas heurísticas. A primeira heurística chama-se de Partial Index Heuristic (PIH) e refere-se ao processo de eleição de índices parciais que, isoladamente, trazem benefícios para a carga de trabalho, corresponde à fase de busca de índices parciais. A implementação desta nova heurística implica estender o DBX com novas estruturas de dados que representam índices parciais e novos meta-dados a serem acrescentados no repositório de meta-dados das consultas submetidas referentes à frequência dos atributos consultados e a inter-relação destes nas consultas. Além disso, acarreta a implementação de técnicas e algoritmos que fazem uso dessas estruturas de dados.

Na implementação da heurística PIH utiliza-se técnicas de compilação para a construção de uma gramática para reconhecer as cláusulas no predicado das consultas e utiliza-se a ferramenta ANTLR (ANother Tool for Language Recognition, [44]) para gerar o código correspondente à gramática na linguajem de programação Java. Além disso, foi acrescentado e modificado o código gerado pela ferramenta para elaborar a árvore de operações lógicas referida na seção 3.1.1. Por outro lado, foram implementadas as estruturas de dados e algoritmos propostos no Capítulo 3.

Por sua vez, a segunda heurística chama-se de Enumeration Heuristic (EH) e corresponde à fase de numeração de configurações de índices parciais e completos em conjunto. A heurística foi elaborada segundo a estratégia de enumeração proposta na seção 3.1.2. As heurísticas PIH e EH são usadas pelo agente Structures Maintenance para desempenhar a seleção de índices parciais e a eleição de configurações de índices parciais e completos. Desse modo, a heurística PIH faz a seleção de aqueles índices parciais que, isoladamente, podem trazer benefícios para a carga de trabalho e a heurística EH recebe estes índices parciais e os índices completos candidatos selecionados na heurística de seleção para índices completos já implementada no DBX e encontra uma configuração destes índices candidatos.

4.2.
Ambiente Computacional

Depois de implementado o procedimento proposto na seção 3.1 é executado um conjunto de testes para determinar se realmente a utilização de índices parciais pode ter como resultado a diminuição das leituras lógicas e o tempo de execução em cargas de trabalho. Para isso, foram geradas e executadas randomicamente cargas de trabalho com a finalidade de observar os resultados gerados pelo DBX.

O cenário em que os testes são desenvolvidos contempla a utilização de um subconjunto de consultas (*vide* Anexos) do *benchmark* TPC-H que possui uma carga de trabalho analítica (OLAP) utilizando o SGBD PostgreSQL na versão 9.4 e o sistema operacional Windows 10 com 64bit, instalado em um computador com processador Quad-Core 1.6Ghz, 4GB de RAM e 100GB de disco rígido.

O referido TPC-H foi duas vezes instanciado para obter dois bancos de dados com 10GB e 100GB, respetivamente. Na Tabela 4.1 e Tabela 4.2 apresentam-se as características obtidas para cada banco de dados gerado. Nota-se que a maior quantidade de tuplas encontra-se nas tabelas *lineitem* e *orders*. Por isso, dependendo da carga de trabalho essas tabelas podem servir de incentivo para uma autossintonia.

Tabela 4-1: Dados das tabelas do *schema* TCP-H (10GB)

Nome	Quantidades de tuplas	Tamanho
Customer	450.000	87 MB
Lineitem	17.996.600	2.848 MB
nation	25	8.192 bytes
Orders	4.500.000	646 MB
Part	600.000	96 MB
Partsupp	2.399.940	427 MB

region	5	8.192 bytes
Supplier	30.000	5.512 bytes

Tabela 4-2: dados das tabelas do *schema* TCP-H (100gb)

Nome	Quantidades de tuplas	Tamanho
Customer	15.000.000	2.799 MB
Lineitem	600.038.000	86 GB
nation	25	8.192 bytes
Orders	150.000.000	20 GB
Part	20.000.000	3.200 MB
Partsupp	80.000.600	13 GB
region	5	8.192 bytes
Supplier	30.000	173 MB

Por outro lado, para a geração das consultas das cargas de trabalho foram escolhidas e parametrizadas oito consultas do *benchmark* TPC-H para servir de base à geração das consultas. As consultas selecionadas do *benchmark* são simples, ou seja, não têm subconsultas aninhadas. Ademais, nas consultas foram parametrizados os atributos tipo *numeric*, l_discount e l_quantity, e os atributos tipo *date*, l_shipdate, o_orderdate e l_receiptdate, e a elaboração das consultas da carga de trabalho foi feita substituindo os parâmetros das consultas com valores randômicos. No caso dos atributos tipo *numeric*, l_discount e l_quantity para a carga de trabalho geram-se valores nos intervalos [0, 1] e [2, 49] que representam 100% das tuplas da tabela lineitem. Para os atributos l_shipdate, o_orderdate e l_receiptdate geram-se valores nos intervalos [1992-01-01, 1993-01-01] e [1997-01-01, 1998-01-01]. No Anexo 1 apresentam-se as consultas parametrizadas.

Com base nas consultas parametrizadas foi gerada uma carga de trabalho com cem consultas, fazendo a escolha das consultas parametrizadas randomicamente segundo a distribuição uniforme. A

seguir, a Tabela 4.3 exibe a quantidade de consultas obtida para cada consulta parametrizada na carga de trabalho.

Tabela 4-3: Quantidade de consultas por consulta parametrizada

	Consultas da carga de trabalho (quantidade**)**
Q1	13
Q2	12
Q3	10
Q4	14
Q5	13
Q6	7
Q7	9
Q8	22

4.3. Resultados dos testes e avaliação

Após a definição do ambiente computacional, desenvolve-se um experimento em duas etapas para comparar os resultados obtidos da execução do DBX com e sem a estratégia de criação de índices parciais para cada carga de trabalho. A primeira etapa consiste na execução das duas cargas de trabalho escolhendo randomicamente uma consulta por vez sem reposição. Além da observação das ações de sintonia fina desenvolvidas pelo DBX configurado com a estratégia de criação de índices parciais. A segunda etapa tem o mesmo procedimento com o DBX configurado com a estratégia de criação de índices parciais.

4.3.1. Primeira fase: DBX sem estratégia de criação de índices parciais

O resultado obtido da execução do DBX para o caso dos índices completos foi idêntico tanto para a carga de trabalho com 10GB como para a carga de trabalho com 100GB. O resultado obtido justifica-se pelo fato de que o DBX não realiza a análise da distribuição dos valores na carga de trabalho. Na Tabela 4.4 mostram-se os índices criados pelo DBX.

Tabela 4-4: Índices criados pelo DBX

Nome do índice	Atributos envolvidos	Tabela
C_4	l_quantity	lineitem
C_4_13_14	l_quantity, l_shipinstruct, l_shipmode	lineitem
C_8	l_returnflag	lineitem
C_10	l_shipdate	lineitem
C_303	p_brand	part
C_104	o_orderdate	orders

A seguir, apresenta-se a Tabela 4.5 dos índices criados por consulta parametrizada.

Tabela 4-5: Índices criados por consulta

Consultas	Índices
Q1	C_10
Q2	C_10
Q3	C_104
Q4	C_10, C_4
Q5	C_8, C_104

Q6	C_12_14
Q7	C_10
Q8	C_4_13_14, C_303

Note-se que todas as consultas parametrizadas da Tabela 4.5 são atendidas por algum índice.

4.3.2. Segunda fase: DBX com estratégia de criação de índices parciais

Nesta fase do experimento executaram-se as cargas de trabalho separadamente do DBX e configurou-se o algoritmo de detecção de padrões frequentes, para considerar conjuntos de atributos frequentes que se encontram mais de 10% (0.10) das transações analisadas. Para a carga de trabalho 1 (10GB) os resultados obtidos foram 26 índices candidatos, dos quais, 15 são índices parciais e 11 índices completos, e foram selecionados na configuração final 8 índices candidatos. Por outro lado, para a carga de trabalho 2 (100GB) foram considerados 25 índices candidatos, destes 10 completos e 15 parciais e definitivamente foram selecionados na configuração final 8 índices candidatos. Nos anexos 2 e 3 observam-se os índices criados pelo DBX nesta segunda fase do experimento para cada carga de trabalho respetivamente.

Na Tabela 4.6 figuram os índices considerados pelo otimizador para cada uma das consultas das cargas de trabalho.

Tabela 4-6: Índices criados para cada consulta parametrizada

Consulta	Carga de trabalho 1 (10GB)	Carga de trabalho 2 (100GB)
Q1	P_10	C_10
Q2	C_10	C_10

Q3	C_104	P_104
Q4	P_4_6_10	P_4_6_10
Q5	P_8, C_104	C_8, P_104
Q6	C_12_14	P_12_14
Q7	P_10	P_10
Q8	P_4_13_14, C_303	P_4_13_14, P_303_306

Com o objetivo de avaliar os resultados obtidos da execução do DBX com a implementação de índices parciais e sem nenhuma modificação foram usadas as medidas de avaliação “quantidade de leituras lógicas” e “tempo de execução das consultas”. A avaliação dos resultados foi feita não só executando as consultas de cada carga de trabalho nas configurações obtidas dos experimentos anteriores, bem como a contagem da quantidade de leituras lógicas e o tempo de execução de cada consulta através do comando “*explain* (*analyze on, buffers on, verbose on*)”. O experimento realizado para pegar o tempo de execução de cada consulta foi executar dez vezes cada consulta rejeitando o primeiro e último resultado.

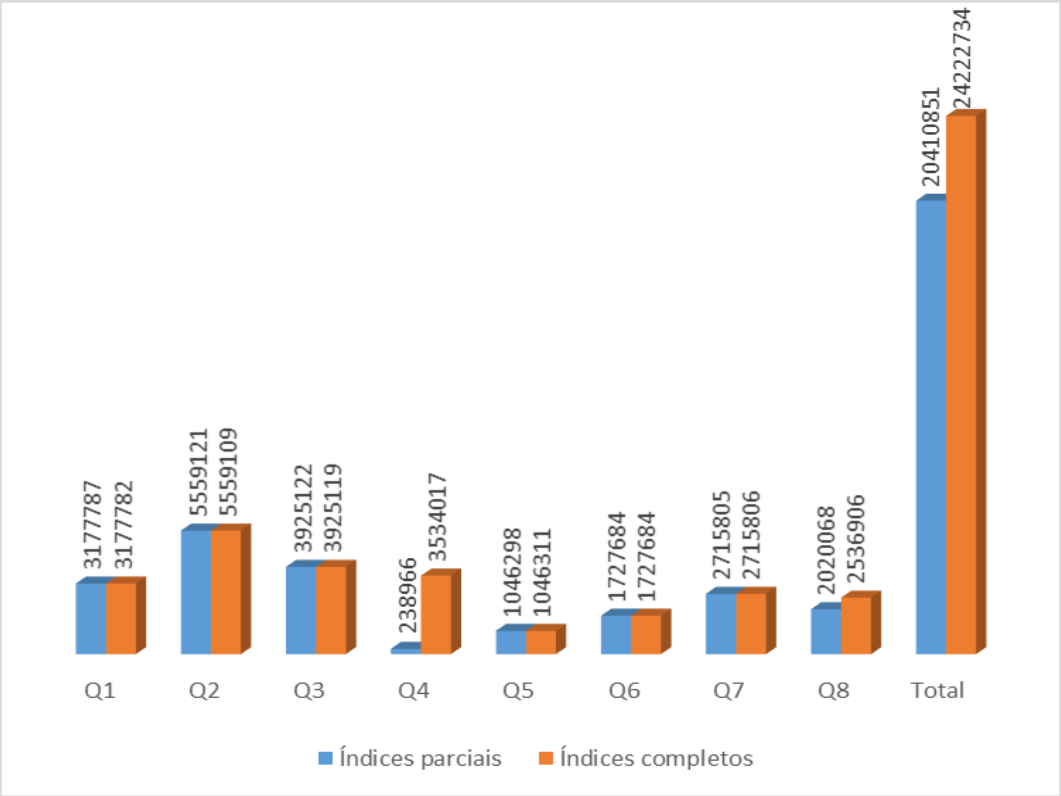


Figura 4-2: Leituras lógicas para a carga de trabalho 1 (10GB)

Na Figura 4.2 observam-se os resultados obtidos conforme as leituras lógicas na execução do DBX com e sem a estratégia de criação de índices parciais para a primeira carga de trabalho em cada consulta parametrizada. O ganho mais significativo ocorreu nas consultas parametrizadas Q4 e Q8 nas quais foram criados índices parciais em várias colunas da tabela, não sendo assim nos índices parciais criados em uma coluna da tabela. No entanto, a quantidade total de leituras lógicas no processamento da carga de trabalho 1 é menor quando os índices são criados usando a estratégia de índices parciais. O que significa que a solução com índices parciais reduziu em cerca de 4.000.000 milhões a quantidade de leituras lógicas.

.

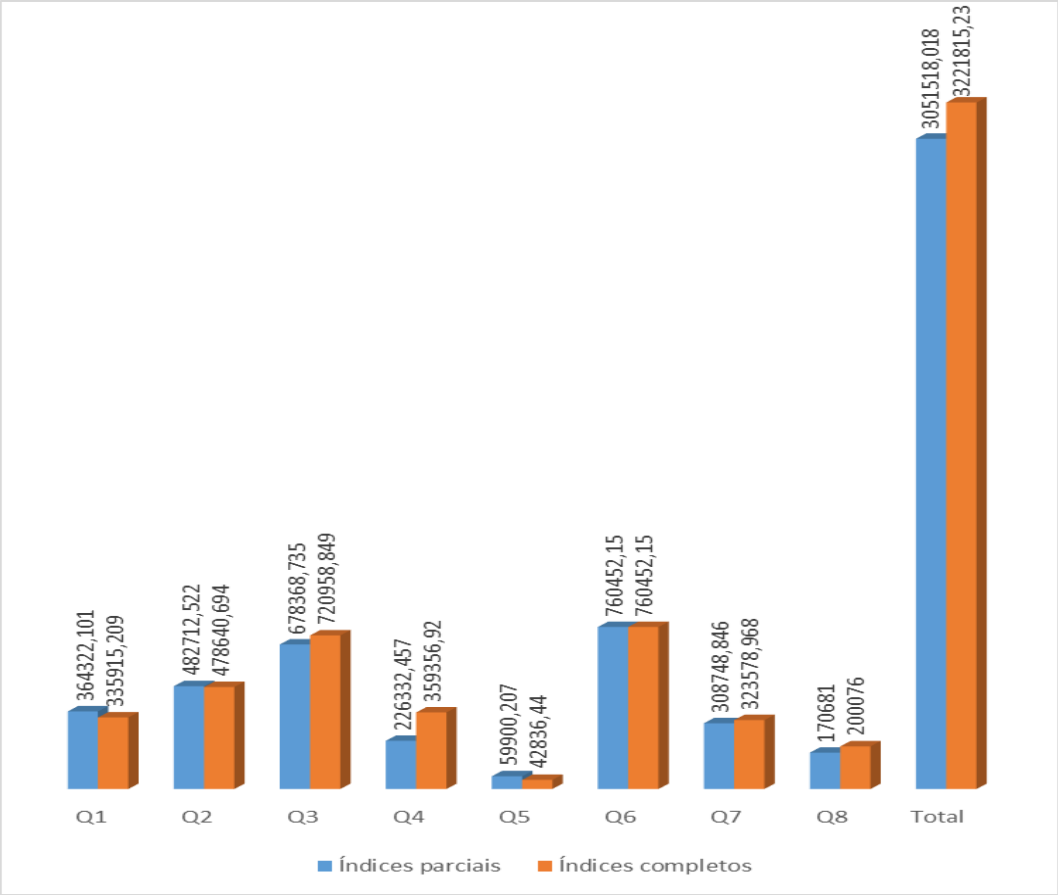


Figura 4-3: Tempo de execução da carga de trabalho 1 (10GB)

Ao passo que na Figura 4.3 apresenta-se o tempo de execução acumulado para cada consulta parametrizada. Neste caso para as consultas parametrizadas Q4 e Q8, como esperado, obtém-se uma grande diminuição no tempo de execução para o caso da estratégia de criação de índices parciais. Nas consultas parametrizadas restantes o tempo de processamento não muda de forma representativa. No caso da carga de trabalho 2, os resultados foram similares aos obtidos para a carga de trabalho 1.

Houve uma diminuição da quantidade de leituras lógicas e, portanto, do tempo de execução nas consultas parametrizadas em que foram utilizados índices parciais. Neste caso, a maior diminuição das leituras lógicas foi nas consultas parametrizadas Q4, Q6 e Q8. Nas Figuras 4.4 e 4.5 são exibidos os resultados obtidos para cada consulta parametrizada segundo as leituras lógicas e os tempos de execução de cada consulta parametrizada. A quantidade de leituras lógicas para a solução com

índices parciais diminuiu cerca de 691.000.000 de leituras lógicas implicando uma redução aproximada de 11 horas de execução.

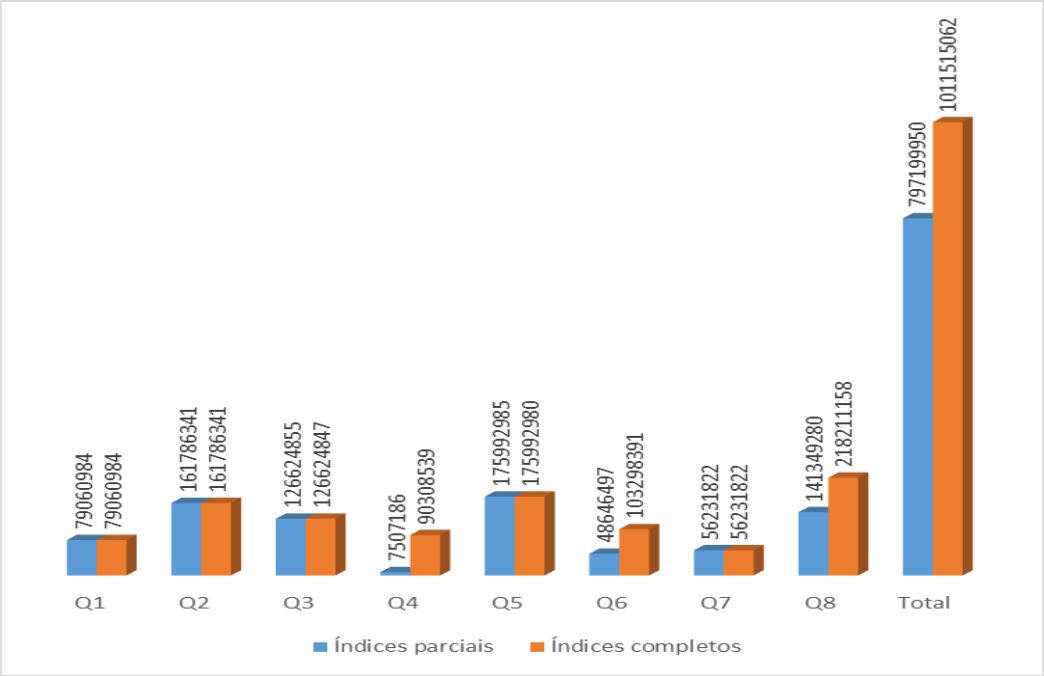


Figura 4-4: Leituras lógicas para a carga de trabalho 2 (100GB)

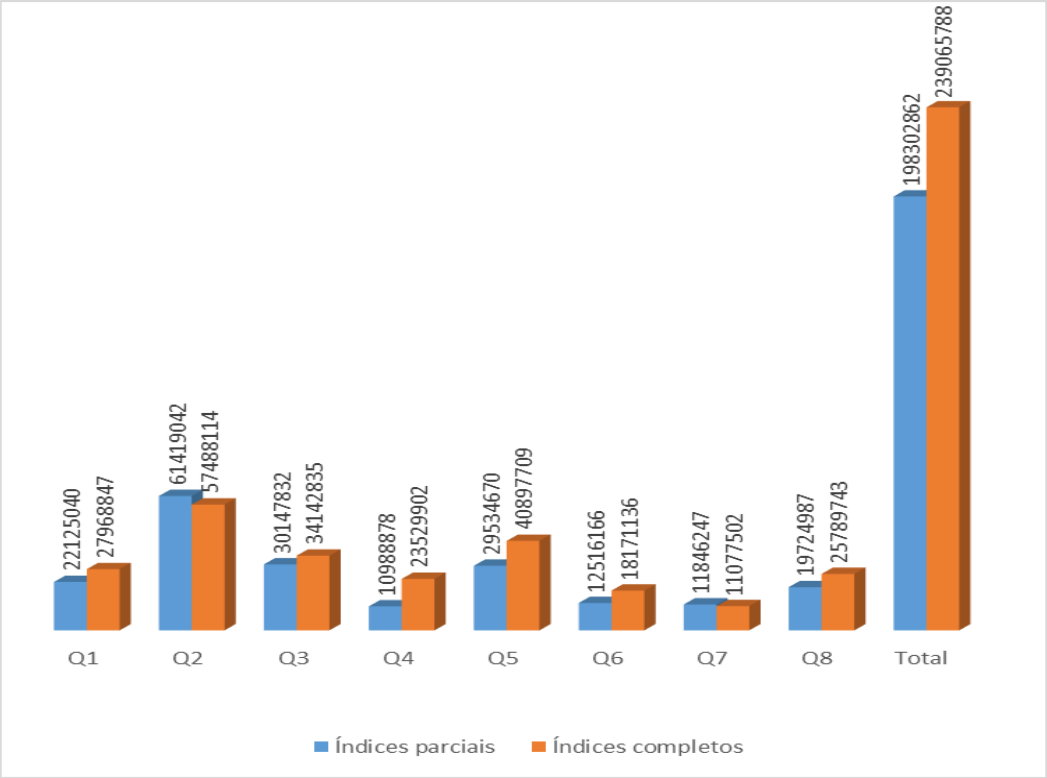


Figura 4-5: Tempo de execução para a carga de trabalho 2 (100GB)

Da análise dos resultados obtidos na avaliação das duas cargas de trabalho observa-se que a maior diminuição das leituras lógicas é experiente nos casos em que se criaram índices parciais em várias colunas de uma tabela. Essa conclusão significa que o comportamento dos índices parciais quando criados em uma coluna da tabela é similar aos índices completos quanto à quantidade de leituras lógicas. Note-se que nesta análise não é considerado que os índices parciais em uma coluna da tabela têm custo de criação menor que os índices completos.

Contudo, no caso dos índices parciais, criados em várias colunas de uma tabela, precisa-se determinar se a diferença no tempo de processamento pode ser considerada significativa. Assim, efetuam-se testes de hipótese para a diferença das médias para determinar se a diferença do tempo de processamento das consultas parametrizadas para as quais foram criados índices parciais de várias colunas pode ser considerada significativa.

A Tabela 4.7 demonstra os resultados obtidos no teste de hipótese com nível de significância de 0.05. Na tabela, *n* representa o tamanho da

amostra; X_1 e S_1 a média e o desvio padrão da amostra na solução com índices parciais; X_2 e S_2 a média e o desvio padrão para a solução com índices completos; e D se a diferença nos tempos de execução foi significativa.

Conforme os resultados dos testes, avaliou-se como significativa a hipótese de que índices parciais criados em várias colunas de uma tabela podem trazer benefícios expressivos para a carga de trabalho.

Tabela 4-7: Teste de hipótese para a diferença da média

Carga de trabalho 1 (10GB)						
Consulta	N	X_1	S_1	X_2	S_2	D
Q4	14	16.167	630	25.668	429	significativa
Q8	22	7.758	228	9.094	354	significativa
Carga de trabalho 2 (100GB)						
Q4	14	784.920	6.282	1.680.707	8.373	significativa
Q6	7	1.788.024	10.371	2.595.876	13.625	significativa
Q8	22	896.590	9.215	1.172.261	9.578	significativa

Por outro lado, a criação de índices abrange o consumo de recursos e de tempo de execução, portanto, o custo de criação dos índices representa um fator influente no processamento da carga de trabalho. Nesta premissa, o tamanho dos índices completos é geralmente maior do que o dos índices parciais, isto se reflete numa maior exigência de memória para alocação do índice e numa maior quantidade de escritas no disco rígido. Assim, o tempo de criação dos índices completos pode ser significativamente maior do que o índice parcial e, por isso, o impacto na carga de trabalho é maior.

A Tabela 4.8 mostra os tempos de execução dos índices completos e o correspondente índice parcial para a carga de trabalho 1 (10GB).

Tabela 4-8: Tempo de criação dos índices da carga de trabalho (10GB)

Carga de trabalho 1 (10GB)			
Índice completo	**Tempo de criação**	**Índice parcial**	**Tempo de criação**
C_10	54 seg	P_10	63 seg
C_12_14	234 seg	P_12_14	55 seg
C_4_13_14	355 seg	P_4_13_14	89 seg
C_4_6_10	164 seg	P_4_6_10	45 seg
C_8	25 seg	P_8	24 seg
C_104	9 seg	P_104	16 seg
C_303	4 seg	P_303	2 seg
Total	845 seg	**Total**	294 seg

Observe que o tempo de criação de índices completos é maior que o tempo para a criação de índices parciais na maioria dos casos. Este resultado torna-se mais evidente no caso da carga de trabalho 2 (100GB) como mostrado na Tabela 4.9 ao evidenciar que a criação de uma configuração de índices completos pode levar cerca de 28 horas de processamento enquanto uma solução de índices parciais 5 horas.

Tabela 4-9: Tempo de criação dos índices da carga de trabalho (100GB)

Carga de trabalho 2 (100GB)			
C_10	53 min 85 seg	P_10	31 min 45 seg
C_8	138 min 75 seg	P_8	30 min 08 seg
C_4_13_14	365 min 09 seg	P_4_13_14	27 min 26 seg
C_4_6_10	158 min 39 seg	P_4_6_10	29 min 87 seg
C_104	9 min 61 seg	P_104	10 min 74 seg
C_303_306	6 min 28 seg	P_303_306	0 min 53 seg
C_12_14	130 min 76 seg	P_12_14	15 min 77 seg
Total	865 min 13 seg	**Total**	148 min 10 seg

4.4. Conclusões

Neste capítulo não só foi implementado o procedimento proposto no Capítulo 3 em uma ferramenta de sintonia fina, bem como fornecido um ambiente computacional para desempenhar testes do procedimento proposto. Estes testes foram realizados em cargas de trabalho criadas randomicamente com um escalonamento do banco de dados de 90GB segundo as medidas de avaliação “quantidade de leituras lógicas” e “tempo de execução”.

Os resultados obtidos dos testes indicam ser possível achar configurações de índices parciais e completos que ajudam a melhorar os tempos de resposta em cargas de trabalho, especialmente aquelas estabelecidas em bancos de dados grandes. Por outro lado, os índices parciais criados em várias colunas de uma tabela apresentam melhores resultados que índices parciais criados em uma coluna. No entanto, o tempo de criação de índices parciais geralmente é menor que o tempo de criação de índices completos, logo, estes índices trazem maior vantagem para a carga de trabalho.

Além disso, pode-se concluir que o processo proposto para criação de índices parciais apresenta resultados relevantes, enquanto os índices parciais derivados do processo têm trazido bons resultados em comparação com a estratégia de criação de índices completos.

5
Considerações Finais

Neste capítulo apresentam-se os principais resultados e repercussões obtidos nesta tese de mestrado, além dos trabalhos futuros a serem desenvolvidos. A principal conclusão do trabalho é que efetivamente os índices parciais são um recurso útil para o processo de sintonia fina em bancos de dados. Neste aspecto, os resultados do trabalho provam que configurações do projeto físico do banco de dados que incluem tanto índices parciais como completos podem resultar em melhorias no desempenho da execução da carga de trabalho.

Além disso, demonstrou-se que o procedimento proposto para a seleção dos índices parciais e completos é válida, ou seja, de fato acharam-se índices parciais benéficos para a carga de trabalho. Demonstrando também que as técnicas e algoritmos apresentados nesta tese servem ao seu propósito.

A completude deste trabalho, que visa na seleção e enumeração de índices parciais em base da carga de trabalho, encontra-se na estratégia de integração de soluções para o caso dos índices parciais e completos. Esta afirmação baseia-se no fato de que o espaço de busca dos índices parciais é reduzido para padrões encontrados nas consultas que determinam conjuntos de dados frequentemente consultados. Isto significa que as consultas que não contêm estes padrões não são atendidas pelos índices parciais e, portanto, precisam de mecanismos para complementar as soluções propostas pela heurística da criação de índices parciais.

5.1.
Contribuições

Nesta pesquisa diferenciam-se os índices parciais e completos, justifica-se porque o otimizador de um SGBD usaria os índices parciais e

os determinou como um importante recurso para ser usado por um DBA. Outra contribuição deste trabalho refere-se à extensão da Heurística de Benefícios com estratégias de análise de frequência que sejam propostas para a criação de índices com impacto sobre uma ou várias consultas. O procedimento proposto conserta técnicas e algoritmos utilizados em outras áreas de pesquisa, desse modo, são usadas técnicas de compilação nas consultas da carga de trabalho para obter conjuntos de atributos com possibilidades de criar índices. Em face disso, a mineração de atributos frequentes é usada para encontrar padrões de consulta usuais na carga de trabalho; o *clustering* é usado nas restrições dos atributos contínuos para diminuir a quantidade de faixas; a heurística de benefícios é usada para obter índices parciais candidatos com benefícios na carga de trabalho; e, finalmente, a estratégia de sintonia fina global é usada para encontrar configurações de tanto índices parciais como completos a serem instaladas no banco de dados.

A metodologia proposta foi implementada e testada obtendo resultados satisfatórios que avaliam o fato de que, na verdade, o procedimento complementa as configurações de índices completos com índices parciais que serão usados pelo otimizador e trazem benefícios no processamento de cargas de trabalho.

5.2.
Trabalho futuros

Elencam-se aqui possíveis oportunidades de pesquisas e os próximos passos na evolução da pesquisa.

- Estender o algoritmo de sintonia fina global para consertar o caso das visões materializadas, índices parciais e completos;
- Estender a interface do DBX para exibir informações sobre os índices parciais candidatos e índices parciais criados;
- Estender a interface do DBX para exibir informações dos atributos frequentemente consultados;

- Estender a experiência adquirida com a aplicação da metodologia implementada em DBX para OuterTuning através da extensão da ontologia para a seleção de índices. Neste caso, tem-se de adicionar o conceito de índice parcial na ontologia de domínio e fazer modificações na ontologia de tarefas [14].

6
Referências Bibliográficas

[1] P. Shasha, D., & Bonnet, **Database tuning: principles, experiments, and troubleshooting techniques**. San Francisco: Morgan Kaufmann, 2002.

[2] Postgresql, “Partial Indexes,” 2016. [Online]. Available: http://www.postgresql.org/docs/9.4/static/indexes-partial.html. [Accessed: 20-Jan-2016].

[3] Microsoft, “Create Filtered Indexes,” 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/cc280372.aspx. [Accessed: 10-Mar-2016].

[4] J. M. da Silva Monteiro, “**Uma Abordagem Não-Intrusiva para a Manutenção Automática do Projeto Físico de Bancos de Dados,**” Phd These, PUC-Rio, 2008.

[5] R. Jones, “Partially-indexed file organization,” 2001. [Online]. Available: http://www.ib-computing.net/program/topic_7/partial.html. [Accessed: 03-Mar-2016].

[6] M. Stonebraker, “**The case for partial indexes,**” ACM SIGMOD Record, vol. 18, no. 4, pp. 4–11, 1989.

[7] TPC, “TPC-H.” [Online]. Available: http://www.tpc.org/tpch/. [Accessed: 03-Mar-2016].

[8] L. T. Kimberly, “How about Filtered Indexes instead of Partitioning?,” 2012. [Online]. Available: http://sqlmag.com/blog/how-about-filtered-indexes-instead-partitioning. [Accessed: 15-Jan-2016].

[9] J. S. Borland, “Filtered Indexes vs. Table Partitioning,” 2013. [Online]. Available: https://www.brentozar.com/archive/2013/11/filtered-indexes-vs-table-partitioning/. [Accessed: 06-Feb-2016].

[10] N. Bruno, S. Chaudhuri, A. Christian König, V. Narasayya, R. Ramamurthy, and M. Syamala, "**AutoAdmin Project at Microsoft Research: Lessons Learned,**" Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, vol. 34, no. 4, pp. 12–19, 2011.

[11] R. Elmasri and S. B. Navathe, **Sistemas de banco de dados**, 6th ed. 2011.

[12] G. Weikum, C. Hasse, A. Mönkeberg, and P. Zabback, "**The COMFORT automatic tuning project,**" Information systems, vol. 19, no. 5, pp. 381–432, 1994.

[13] S. Lifschitz, A. Y. Milanés, and M. A. V. Salles, "**Estado da Arte em Auto-sintonia de SGBD Relacionais,**" PUC-Rio, Technical Report, 2004.

[14] A. Carolina Brito De Almeida, "**Framework para apoiar a sintonia fina de banco de dados,**" Phd Thesis, PUC-Rio, 2013.

[15] S. Chaudhuri and V. R. Narasayya, "**An efficient, cost-driven index selection tool for Microsoft SQL server,**" in VLDB, 1997, vol. 97, pp. 146–155.

[16] J. M. Monteiro, S. Lifschitz, and Â. Brayner, "**Automated Selection of Materialized Views Automated Selection of Materialized Views *,**" PUC-Rio, Technical Report, 2006.

[17] G. Valentin, M. Zuliani, D. C. Zilio, G. Lohman, and A. Skelley, "**DB2 advisor**: N" in ICDE, 2000, p. 101.

[18] R. L. De Carvalho Costa, S. Lifschitz, M. F. De Noronha, and M. A. V. Salles, "**Implementation of an agent architecture for automated index tuning,**" In Data Engineering Workshops, 2005. 21st International Conference on (pp. 1215-1215), vol. 2005, 2005.

[19] P. Cudre-Mauroux, E. Wu, and S. Madden, "**Trajstore: An adaptive storage system for very large trajectory data sets,**" in Data

Engineering (ICDE), 2010 IEEE 26th International Conference on, 2010, pp. 109–120.

[20] E. Wu and S. Madden, “**Partitioning techniques for fine-grained indexing,**” Proceedings - International Conference on Data Engineering, pp. 1127–1138, 2011.

[21] P. Seshadri and A. Swami, “**Generalized Partial Indexes,**” Proceedings of The 11th International Conference on Data Engineering, 1995., pp. 420–427, 1995.

[22] S. Wu, J. Li, B. C. Ooi, and K.-L. Tan, “**Just-in-time query retrieval over partially indexed data on structured P2P overlays,**” Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD, p. 279, 2008.

[23] P. Lu, S. Wu, L. Shou, and K. L. Tan, “**An efficient and compact indexing scheme for large-scale data store,**” Proceedings - International Conference on Data Engineering, pp. 326–337, 2013.

[24] M. L. Kersten, S. Manegold, and others, “**Cracking the database store,**” in CIDR, 2005, vol. 5, pp. 4–7.

[25] S. Idreos, M. Kersten, and S. Manegold, “**Database Cracking,**” CIDR ’07: 3rd Biennial Conference on Innovative Data Systems Research, pp. 68–78, 2007.

[26] H. Voigt, T. Kissinger, and W. Lehner, “**SMIX: Self-Managing Indexes for Dynamic Workloads,**” Proceedings of the 25th International Conference on Scientific and Statistical Database Management - SSDBM, p. 1, 2013.

[27] G. Graefe and H. Kuno, “**Self-selecting, self-tuning, incrementally optimized indexes,**” Proceedings of the 13th International Conference on Extending Database Technology - EDBT ’10, p. 371, 2010.

[28] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, “**Efficient mining**

of association rules using closed itemset lattices," Information Systems, vol. 24, no. 1, pp. 25–46, 1999.

[29] K. Gouda and M. Zaki, "**Efficiently mining maximal frequent itemsets,**" in Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on, 2001, pp. 163–170.

[30] R. L. De Carvalho Costa, S. Lifschitz, and M. A. V. Salles, "**Index self-tuning with agent-based databases,**" CLEI Electronic Journal, vol. 6, no. 1, 2003.

[31] M. A. . Salles, "**Criação autônoma de Índices em Bancos de Dados,**" MSc These, PUC-Rio, 2004.

[32] E. Maria Terra Morelli and S. Lifschitz, "**Recriaçao Automática de Índices em um SGBD Relacional,**" MSc These, PUC-Rio, 2006.

[33] A. W. Carvalho, "**Criação Automática de Visões Materializadas em SGBDs Relacionais,**" phdthesis, PUC-Rio, 2011.

[34] R. Pereira de Oliveira, "**Sintonia fina baseada em ontologia: o caso de visoes materializadas,**" MSc These, PUC-Rio, 2015.

[35] IBM, "**Predicate processing for queries,**" IBM Knowledge Center, 2015.

[36] S. Azefack, K. Aouiche, and J. Darmont, "**Dynamic index selection in data warehouses,**" in Innovations in Information Technology, 2007. IIT'07. 4th International Conference on, 2007, pp. 1–5.

[37] K. Gouda and M. J. Zaki, "**GenMax: An efficient algorithm for mining maximal frequent itemsets,**" Data Mining and Knowledge Discovery, vol. 11, no. 3, pp. 223–242, 2005.

[38] P. Valtchev, R. Missaoui, R. Godin, and M. Meridji, "**Generating frequent itemsets incrementally: two novel approaches based on Galois lattice theory,**" Journal of Experimental & Theoretical Artificial Intelligence, vol. 14, no. 2–3, pp. 115–142, 2002.

[39] D. Burdick, M. Calimlim, and J. Gehrke, "**MAFIA: A maximal frequent itemset algorithm for transactional databases,**" Proceedings of the 17th IEEE International Conference on Data Engineering (ICDE), pp. 443–452, 2001.

[40] R. De La Briandais, "**File searching using variable length keys,**" in Western joint computer conference. Papers presented at the the March 3-5, 1959, pp. 295–298.

[41] E. Fredkin, "**Trie memory,**" Communications of the ACM, vol. 3, no. 9, pp. 490–499, 1960.

[42] S. Asharaf, M. N. Murty, and S. K. Shevade, "**Rough set based incremental clustering of interval data,**" Pattern Recognition Letters, vol. 27, no. 6, pp. 515–519, 2006.

[43] K. Aouiche and J. Darmont, "**Data mining-based materialized view and index selection in data warehouses,**" Journal of Intelligent Information Systems, vol. 33, no. 1, pp. 65–93, 2009.

[44] T. Parr, "ANTLR," © Copyright ANTLR / Terence Parr, 2014. [Online]. Available: http://www.antlr.org/. [Accessed: 05-Nov-2015].

Anexo 1: Consultas parametrizadas

Consulta 1

```
select l_returnflag, l_linestatus, sum(l_quantity) as sum_qty,
 sum(l_extendedprice) as sum_base_price,
 sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
 sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
 avg(l_quantity) as avg_qty, avg(l_extendedprice) as avg_price,
 avg(l_discount) as avg_disc, count(*) as count_order
from lineitem
where
 l_shipdate > date ?VFDL0 and
 l_shipdate < date ?VFDG0
group by l_returnflag, l_linestatus
order by l_returnflag, l_linestatus
```

Consulta 2

```
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 ?VFDL0 and
 l_shipdate > date ?VFDG0
group by l_orderkey, o_orderdate, o_shippriority
order by revenue desc, o_orderdate
```

Consulta 3

```
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 = 'AFRICA ' and
 o_orderdate > date ?VFDL0 and
 o_orderdate < date ?VFDG0
group by n_name
order by revenue desc
```

Consulta 4

```
select sum(l_extendedprice * l_discount) as revenue
from lineitem
where l_shipdate > date ?VFDL0 and
 l_shipdate < date ?VFDG0 and
 l_discount < ?VFNG0 and
 l_discount > ?VFNL0 and
 l_quantity < ?VFNG1
```

Consulta 5

```
select c_custkey, c_name, sum(l_extendedprice * (1 - l_discount)) as revenue,
c_acctbal, n_name, c_address, c_phone, c_comment
from customer, orders, lineitem, nation
where c_custkey = o_custkey and
 l_orderkey = o_orderkey and
 o_orderdate > date ?VFDL0 and
 o_orderdate < date ?VFDG0 and
 l_returnflag = 'A' and
 c_nationkey = n_nationkey
group by c_custkey, c_name, c_acctbal, c_phone, n_name, c_address,
c_comment
order by revenue desc
```

Consulta 6

```
select l_shipmode, sum(case when o_orderpriority = '1-URGENT' or
o_orderpriority = '2-HIGH' then 1 else 0 end) as high_line_count,
sum(case when o_orderpriority <> '1-URGENT' and o_orderpriority <> '2-HIGH'
then 1 else 0 end) as low_line_count
from orders, lineitem
where
 o_orderkey = l_orderkey and
 (l_shipmode = 'REG AIR ' or l_shipmode = 'AIR ') and
 l_commitdate < l_receiptdate and
 l_shipdate < l_commitdate and
 l_receiptdate > date ?VFDL0 and
 l_receiptdate < date ?VFDG0
group by l_shipmode
order by l_shipmode
```

Consulta 7

```
select 100.00 * sum(case when p_type like 'PROMO%' then l_extendedprice * (1
- l_discount) else 0 end) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue
from lineitem, part
where l_partkey = p_partkey and
 l_shipdate > date ?VFDL0 and
 l_shipdate < date ?VFDG0
```

Consulta 8

```
select sum(l_extendedprice* (1 - l_discount)) as revenue
from lineitem, part
where
 (
 p_partkey = l_partkey and
 p_brand = 'Brand#14 ' and
 (p_container ='SM CASE' or p_container = 'SM BOX' or p_container ='SM PACK'
or p_container ='SM PKG') and
 l_quantity > ?VFNL0 and l_quantity < ?VFNG0 and
 (l_shipmode ='AIR' or l_shipmode ='AIR REG') and
 l_shipinstruct = 'DELIVER IN PERSON'
 )
 or
 (
 p_partkey = l_partkey and
 p_brand = 'Brand#14 ' and
 (p_container ='MED BAG' or p_container ='MED BOX' or p_container = 'MED
PKG' or p_container = 'MED PACK') and
 l_quantity > ?VFNL1 and l_quantity < ?VFNG1 and
 (l_shipmode ='AIR' or l_shipmode ='AIR REG') and
 l_shipinstruct = 'DELIVER IN PERSON'
)
```

Anexo 2: Índices criados para a carga de trabalho 1 (10GB)

```
CREATE INDEX c_10
ON lineitem
USING btree
(l_shipdate);


CREATE INDEX p_10
ON lineitem
USING btree
(l_shipdate)
WHERE
l_shipdate < '1997-11-07'::date AND l_shipdate > '1997-07-19'::date
 OR
l_shipdate < '1997-12-28'::date AND l_shipdate > '1997-12-12'::date
 OR
l_shipdate < '1997-12-21'::date AND l_shipdate > '1997-02-11'::date
 OR
l_shipdate > '1997-12-18'::date
 OR
l_shipdate < '1997-09-19'::date AND l_shipdate > '1997-03-01'::date
 OR
l_shipdate < '1997-12-10'::date AND l_shipdate > '1997-10-27'::date
 OR
l_shipdate < '1997-09-20'::date AND l_shipdate > '1997-05-05'::date
 OR
l_shipdate < '1997-11-14'::date AND l_shipdate > '1997-06-23'::date
 OR
l_shipdate > '1997-07-28'::date OR l_shipdate > '1997-10-02'::date
 OR
l_shipdate > '1997-10-07'::date
 OR
l_shipdate < '1997-11-21'::date AND l_shipdate > '1997-09-18'::date
 OR
l_shipdate < '1997-11-13'::date AND l_shipdate > '1997-09-22'::date
 OR
l_shipdate < '1997-08-17'::date AND l_shipdate > '1997-02-27'::date
 OR
l_shipdate < '1997-09-02'::date AND l_shipdate > '1997-02-11'::date
 OR
l_shipdate < '1997-11-19'::date AND l_shipdate > '1997-03-12'::date
 OR
l_shipdate > '1997-12-29'::date OR l_shipdate > '1997-05-11'::date
 OR
l_shipdate < '1997-11-04'::date AND l_shipdate > '1997-10-26'::date
 OR
l_shipdate < '1997-11-28'::date AND l_shipdate > '1997-03-14'::date
 OR
l_shipdate < '1997-12-29'::date AND l_shipdate > '1997-11-06'::date
 OR
l_shipdate < '1997-12-30'::date AND l_shipdate > '1997-08-12'::date
 OR
l_shipdate > '1997-09-08'::date OR l_shipdate > '1997-03-24'::date
 OR
l_shipdate < '1997-12-15'::date AND l_shipdate > '1997-06-11'::date
 OR
```

```
        l_shipdate < '1997-09-18'::date AND l_shipdate > '1997-01-18'::date
         OR
        l_shipdate < '1997-03-10'::date AND l_shipdate > '1997-02-04'::date
         OR
        l_shipdate < '1997-12-15'::date AND l_shipdate > '1997-09-21'::date
         OR
        l_shipdate > '1997-11-03'::date
         OR
        l_shipdate < '1997-12-15'::date AND l_shipdate > '1997-06-22'::date
         OR
        l_shipdate > '1997-11-22'::date
         OR
        l_shipdate < '1997-07-05'::date AND l_shipdate > '1997-05-12'::date
         OR
        l_shipdate < '1997-11-30'::date AND l_shipdate > '1997-03-05'::date
         OR
        l_shipdate > '1997-10-12'::date OR l_shipdate > '1997-11-08'::date
         OR
        l_shipdate < '1997-12-24'::date AND l_shipdate > '1997-12-15'::date
         OR
        l_shipdate < '1997-08-13'::date AND l_shipdate > '1997-05-08'::date
         OR
        l_shipdate < '1997-10-03'::date AND l_shipdate > '1997-03-30'::date
         OR
        l_shipdate < '1997-11-13'::date AND l_shipdate > '1997-05-17'::date;


        CREATE INDEX c_12_14
        ON lineitem
        USING btree
        (l_shipmode COLLATE pg_catalog."default", l_receiptdate);



        CREATE INDEX p_4_13_14
         ON lineitem
         USING btree
         (l_quantity, l_shipinstruct COLLATE pg_catalog."default", l_shipmode COLLATE
pg_catalog."default")
         WHERE ( L_QUANTITY < 13.058086375359533 AND L_QUANTITY >
7.685234353397091 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR' )
         OR
        ( L_QUANTITY < 13.058086375359533 AND L_QUANTITY >
7.685234353397091 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR REG' )
         OR
        ( L_QUANTITY < 13.074565818623025 AND L_QUANTITY >
4.4431508475628565 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR' )
         OR
        ( L_QUANTITY < 13.074565818623025 AND L_QUANTITY >
4.4431508475628565 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR REG' )
         OR
        ( L_QUANTITY < 13.144078204270654 AND L_QUANTITY >
3.8326392129350366 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR' )
         OR
```

```
        ( L_QUANTITY < 13.144078204270654 AND L_QUANTITY >
3.8326392129350366 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR REG' )
        OR
        ( L_QUANTITY < 17.729294405193944 AND L_QUANTITY >
6.400992340943417 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR' )
        OR
        ( L_QUANTITY < 17.729294405193944 AND L_QUANTITY >
6.400992340943417 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR REG' )
        OR
        ( L_QUANTITY < 19.726627575435277 AND L_QUANTITY >
11.554557392838685 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR' )
        OR
        ( L_QUANTITY < 19.726627575435277 AND L_QUANTITY >
11.554557392838685 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR REG' )
        OR
        ( L_QUANTITY < 19.973747493757152 AND L_QUANTITY >
19.42334892959405 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR' )
        OR
        ( L_QUANTITY < 19.973747493757152 AND L_QUANTITY >
19.42334892959405 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR REG' )
        OR
        ( L_QUANTITY < 20.363782209224638 AND L_QUANTITY >
8.592161900463697 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR' )
        OR
        ( L_QUANTITY < 20.363782209224638 AND L_QUANTITY >
8.592161900463697 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR REG' )
        OR
        ( L_QUANTITY < 21.013180222097482 AND L_QUANTITY >
11.855293459925926 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR' )
        OR
        ( L_QUANTITY < 21.013180222097482 AND L_QUANTITY >
11.855293459925926 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR REG' )
        OR
        ( L_QUANTITY < 21.277760728639628 AND L_QUANTITY >
17.16324739390485 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR' )
        OR
        ( L_QUANTITY < 21.277760728639628 AND L_QUANTITY >
17.16324739390485 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR REG' )
        OR
        ( L_QUANTITY < 21.612358918648976 AND L_QUANTITY >
5.205766389851035 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR' )
        OR
        ( L_QUANTITY < 21.612358918648976 AND L_QUANTITY >
5.205766389851035 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR REG' )
        OR
```

```
        ( L_QUANTITY < 21.721417441546066 AND L_QUANTITY >
17.06519128474062 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR' )
        OR
        ( L_QUANTITY < 21.721417441546066 AND L_QUANTITY >
17.06519128474062 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR REG' )
        OR
        ( L_QUANTITY < 21.804200351221777 AND L_QUANTITY >
20.754572350293657 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR' )
        OR
        ( L_QUANTITY < 21.804200351221777 AND L_QUANTITY >
20.754572350293657 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR REG' )
        OR
        ( L_QUANTITY < 21.826561729161746 AND L_SHIPINSTRUCT = 'DELIVER IN
PERSON' AND L_SHIPMODE = 'AIR' ) OR
        ( L_QUANTITY < 21.826561729161746 AND L_SHIPINSTRUCT = 'DELIVER IN
PERSON' AND L_SHIPMODE = 'AIR REG' ) OR
        ( L_QUANTITY < 21.829427294522556 AND L_QUANTITY >
19.009811124213808 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR' )
        OR
        ( L_QUANTITY < 21.829427294522556 AND L_QUANTITY >
19.009811124213808 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR REG' )
        OR
        ( L_QUANTITY < 21.88619808294939 AND L_QUANTITY >
21.530049003123395 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR' )
        OR
        ( L_QUANTITY < 21.88619808294939 AND L_QUANTITY >
21.530049003123395 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR REG' )
        OR
        ( L_QUANTITY < 21.88697710685508 AND L_SHIPINSTRUCT = 'DELIVER IN
PERSON' AND L_SHIPMODE = 'AIR' )
        OR
        ( L_QUANTITY < 21.88697710685508 AND L_SHIPINSTRUCT = 'DELIVER IN
PERSON' AND L_SHIPMODE = 'AIR REG' ) OR
        ( L_QUANTITY < 21.88697710685508 AND L_QUANTITY >
21.606634082861444 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR' )
        OR
        ( L_QUANTITY < 21.88697710685508 AND L_QUANTITY >
21.606634082861444 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR REG' )
        OR
        ( L_QUANTITY < 6.537656193382689 AND L_QUANTITY >
2.6506868182323378 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR' )
        OR
        ( L_QUANTITY < 6.537656193382689 AND L_QUANTITY >
2.6506868182323378 AND L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND
L_SHIPMODE = 'AIR REG' )
        ;

        CREATE INDEX p_4_6_10
        ON lineitem
```

```
        USING btree
        (l_quantity, l_discount, l_shipdate)
        WHERE
        l_quantity < 18.2811003923472 AND l_discount < 0.023341357999829684 AND
l_discount > 0.018050010074103496 AND l_shipdate < '1997-08-21'::date AND
l_shipdate > '1997-08-12'::date
        OR
        l_quantity < 21.666989134568418 AND l_discount < 0.004121353997759745 AND
l_discount > 0.0038651513156768855 AND l_shipdate < '1997-11-07'::date AND
l_shipdate > '1997-07-19'::date
        OR
        l_quantity < 21.826561729161746 AND l_discount < 0.02689858944060175 AND
l_discount > 0.0038651513156768855 AND l_shipdate > '1997-02-11'::date
        OR
        l_quantity < 21.826561729161746 AND l_discount < 0.02689858944060175 AND
l_discount > 0.007071355868115796 AND l_shipdate < '1997-12-21'::date AND
l_shipdate > '1997-02-11'::date
        OR
        l_quantity < 15.702815436753523 AND l_discount < 0.0245999444624751 AND
l_discount > 0.01676940262235773 AND l_shipdate < '1997-12-10'::date AND
l_shipdate > '1997-10-27'::date
        OR
        l_quantity < 21.826561729161746 AND l_discount < 0.02689858944060175 AND
l_discount > 0.012656307650684718 AND l_shipdate < '1997-12-21'::date AND
l_shipdate > '1997-02-11'::date
        OR
        l_quantity < 21.051974780381435 AND l_discount < 0.011978658096637933 AND
l_discount > 0.008760678573503338 AND l_shipdate < '1997-09-02'::date AND
l_shipdate > '1997-02-11'::date
        OR
        l_quantity < 21.826561729161746 AND l_discount < 0.02834339277229088 AND
l_discount > 0.0038651513156768855 AND l_shipdate > '1997-02-11'::date
        OR
        l_quantity < 21.035837899903893 AND l_discount < 0.02834339277229088 AND
l_discount > 0.024092944232228637 AND l_shipdate < '1997-11-19'::date AND
l_shipdate > '1997-03-12'::date
        OR
        l_quantity < 9.020350886249453 AND l_discount < 0.017931561233652735 AND
l_discount > 0.007939723121103179 AND l_shipdate < '1997-11-04'::date AND
l_shipdate > '1997-10-26'::date
        OR
        l_quantity < 21.88697710685508 AND l_discount < 0.02840772987973926 AND
l_discount > 0.0038651513156768855 AND l_shipdate > '1997-01-18'::date
        OR
        l_quantity < 9.772429999364398 AND l_discount < 0.02636983897058536 AND
l_discount > 0.013967884488129835 AND l_shipdate < '1997-11-01'::date AND
l_shipdate > '1997-10-08'::date
        OR
        l_quantity < 21.88697710685508 AND l_discount < 0.02840772987973926 AND
l_discount > 0.0016128480368708886 AND l_shipdate > '1997-01-18'::date
        OR
        l_quantity < 18.26829224583729 AND l_discount < 0.0032273764692443366 AND
l_discount > 0.0016128480368708886 AND l_shipdate < '1997-11-30'::date AND
l_shipdate > '1997-03-05'::date
        OR
        l_quantity < 11.342758059655893 AND l_discount < 0.02134765826100224 AND
l_discount > 0.018397866843956345 AND l_shipdate < '1997-08-13'::date AND
l_shipdate > '1997-05-08'::date
        OR
```

```
        l_quantity < 21.70198717123795 AND l_discount < 0.021197833591757434 AND
l_discount > 0.013101787594299355 AND l_shipdate < '1997-11-13'::date AND
l_shipdate > '1997-05-17'::date
         OR
        l_quantity < 21.750792226504927 AND l_discount < 0.02054023061522453 AND
l_discount > 0.01766455125443555 AND l_shipdate < '1997-08-17'::date AND
l_shipdate > '1997-02-27'::date
         OR
        l_quantity < 21.826561729161746 AND l_discount < 0.02840772987973926 AND
l_discount > 0.0038651513156768855 AND l_shipdate > '1997-02-11'::date
         OR l_quantity < 8.006048062149967 AND l_discount < 0.02840772987973926
AND l_discount > 0.006111926262827921 AND l_shipdate < '1997-12-29'::date AND
l_shipdate > '1997-11-06'::date;


        CREATE INDEX p_8
        ON lineitem
        USING btree
        (l_returnflag COLLATE pg_catalog."default")
        WHERE l_returnflag = 'A'::bpchar;


        CREATE INDEX c_104
        ON orders
        USING btree
        (o_orderdate);

        CREATE INDEX c_303
        ON part
        USING btree
        (p_brand COLLATE pg_catalog."default");
```

Anexo 3: Índices criados para a carga de trabalho 1 (100GB)

```
CREATE INDEX c_10
 ON lineitem
 USING btree
 (l_shipdate);

CREATE INDEX c_8
 ON lineitem
 USING btree
 (l_returnflag COLLATE pg_catalog."default");

CREATE INDEX p_10
 ON lineitem
 USING btree
 (l_shipdate)
 WHERE l_shipdate < '1997-09-02'::date AND l_shipdate > '1997-02-11'::date
 OR
 l_shipdate < '1997-09-19'::date AND l_shipdate > '1997-03-01'::date
 OR
 l_shipdate < '1997-10-05'::date AND l_shipdate > '1997-07-28'::date
 OR l_shipdate < '1997-10-14'::date AND l_shipdate > '1997-10-09'::date
 OR
 l_shipdate < '1997-11-07'::date AND l_shipdate > '1997-07-19'::date
 OR
 l_shipdate < '1997-11-14'::date AND l_shipdate > '1997-06-23'::date
 OR
 l_shipdate < '1997-11-21'::date AND l_shipdate > '1997-09-18'::date
 OR
 l_shipdate < '1997-12-29'::date AND l_shipdate > '1997-11-06'::date
 OR
 l_shipdate > '1997-03-24'::date OR l_shipdate > '1997-06-11'::date
 OR
 l_shipdate < '1997-11-30'::date AND l_shipdate > '1997-03-05'::date
 OR
 l_shipdate < '1997-12-10'::date AND l_shipdate > '1997-10-27'::date
 OR
 l_shipdate < '1997-12-21'::date AND l_shipdate > '1997-10-02'::date
 OR
 l_shipdate < '1997-11-13'::date AND l_shipdate > '1997-05-17'::date
 OR
 l_shipdate < '1997-03-10'::date AND l_shipdate > '1997-02-04'::date
 OR
 l_shipdate < '1997-09-18'::date AND l_shipdate > '1997-01-18'::date
 OR
 l_shipdate < '1997-12-15'::date AND l_shipdate > '1997-06-22'::date
 OR
 l_shipdate < '1997-12-30'::date AND l_shipdate > '1997-08-12'::date
 OR
 l_shipdate < '1997-06-11'::date AND l_shipdate > '1997-02-11'::date
 OR
 l_shipdate < '1997-07-05'::date AND l_shipdate > '1997-05-12'::date
 OR
 l_shipdate < '1997-08-09'::date AND l_shipdate > '1997-06-02'::date
 OR
 l_shipdate < '1997-08-17'::date AND l_shipdate > '1997-02-27'::date
 OR
 l_shipdate < '1997-09-02'::date AND l_shipdate > '1997-07-13'::date
 OR
 l_shipdate > '1997-05-11'::date;
```

```
        CREATE INDEX p_4_13_14
         ON lineitem
         USING btree
         (l_quantity, l_shipinstruct COLLATE pg_catalog."default", l_shipmode COLLATE
pg_catalog."default")
         WHERE l_quantity < 13.144078204270654 AND l_quantity >
3.8326392129350366 AND l_shipinstruct = 'DELIVER IN PERSON'::bpchar AND
l_shipmode = 'AIR'::bpchar OR l_quantity < 13.144078204270654 AND l_quantity >
3.8326392129350366 AND l_shipinstruct = 'DELIVER IN PERSON'::bpchar AND
l_shipmode = 'AIR REG'::bpchar OR l_quantity < 18.83615584074578 AND l_quantity >
4.217496800557237 AND l_shipinstruct = 'DELIVER IN PERSON'::bpchar AND
l_shipmode = 'AIR'::bpchar OR l_quantity < 18.83615584074578 AND l_quantity >
4.217496800557237 AND l_shipinstruct = 'DELIVER IN PERSON'::bpchar AND
l_shipmode = 'AIR REG'::bpchar OR l_quantity < 18.900223499571077 AND l_quantity >
7.729150561354452 AND l_shipinstruct = 'DELIVER IN PERSON'::bpchar AND
l_shipmode = 'AIR'::bpchar OR l_quantity < 18.900223499571077 AND l_quantity >
7.729150561354452 AND l_shipinstruct = 'DELIVER IN PERSON'::bpchar AND
l_shipmode = 'AIR REG'::bpchar OR l_quantity < 20.363782209224638 AND l_quantity >
8.592161900463697 AND l_shipinstruct = 'DELIVER IN PERSON'::bpchar AND
l_shipmode = 'AIR'::bpchar OR l_quantity < 20.363782209224638 AND l_quantity >
8.592161900463697 AND l_shipinstruct = 'DELIVER IN PERSON'::bpchar AND
l_shipmode = 'AIR REG'::bpchar OR l_quantity < 21.612358918648976 AND l_quantity >
5.205766389851035 AND l_shipinstruct = 'DELIVER IN PERSON'::bpchar AND
l_shipmode = 'AIR'::bpchar OR l_quantity < 21.612358918648976 AND l_quantity >
5.205766389851035 AND l_shipinstruct = 'DELIVER IN PERSON'::bpchar AND
l_shipmode = 'AIR REG'::bpchar OR l_quantity < 21.88697710685508 AND l_shipinstruct
= 'DELIVER IN PERSON'::bpchar AND l_shipmode = 'AIR'::bpchar OR l_quantity <
21.88697710685508 AND l_shipinstruct = 'DELIVER IN PERSON'::bpchar AND
l_shipmode = 'AIR REG'::bpchar OR l_quantity < 21.88697710685508 AND l_quantity >
9.512009501171956 AND l_shipinstruct = 'DELIVER IN PERSON'::bpchar AND
l_shipmode = 'AIR'::bpchar OR l_quantity < 21.88697710685508 AND l_quantity >
9.512009501171956 AND l_shipinstruct = 'DELIVER IN PERSON'::bpchar AND
l_shipmode = 'AIR REG'::bpchar OR l_quantity < 6.537656193382689 AND l_quantity >
2.6506868182323378 AND l_shipinstruct = 'DELIVER IN PERSON'::bpchar AND
l_shipmode = 'AIR'::bpchar OR l_quantity < 6.537656193382689 AND l_quantity >
2.6506868182323378 AND l_shipinstruct = 'DELIVER IN PERSON'::bpchar AND
l_shipmode = 'AIR REG'::bpchar;

        CREATE INDEX p_12_14
         ON lineitem
         USING btree
         (l_receiptdate, l_shipmode COLLATE pg_catalog."default")
         WHERE l_receiptdate > '1997-01-08'::date AND l_receiptdate < '1997-12-
16'::date AND l_shipmode = 'AIR '::bpchar OR l_receiptdate > '1997-01-08'::date AND
l_receiptdate < '1997-12-16'::date AND l_shipmode = 'REG AIR '::bpchar OR
l_receiptdate > '1997-06-28'::date AND l_receiptdate < '1997-12-16'::date AND
l_shipmode = 'AIR '::bpchar OR l_receiptdate > '1997-06-28'::date AND l_receiptdate <
'1997-12-16'::date AND l_shipmode = 'REG AIR '::bpchar;


        CREATE INDEX p_4_6_10
         ON lineitem
         USING btree
         (l_discount, l_quantity, l_shipdate)
         WHERE l_quantity < 20.816960545050463 AND l_discount <
0.02689858944060175 AND l_discount > 0.012656307650684718 AND l_shipdate <
'1997-06-11'::date AND l_shipdate > '1997-02-11'::date
         OR
```

```
        l_quantity < 21.051974780381435 AND l_discount < 0.011978658096637933
AND l_discount > 0.008760678573503338 AND l_shipdate < '1997-09-02'::date AND
l_shipdate > '1997-02-11'::date
        OR
        l_quantity < 21.666989134568418 AND l_discount < 0.004121353997759745
AND l_discount > 0.0038651513156768855 AND l_shipdate < '1997-11-07'::date AND
l_shipdate > '1997-07-19'::date
        OR
        l_quantity < 21.70198717123795 AND l_discount < 0.021197833591757434 AND
l_discount > 0.013101787594299355 AND l_shipdate < '1997-11-13'::date AND
l_shipdate > '1997-05-17'::date
        OR
        l_quantity < 21.750792226504927 AND l_discount < 0.02054023061522453 AND
l_discount > 0.01766455125443555 AND l_shipdate < '1997-08-17'::date AND
l_shipdate > '1997-02-27'::date
        OR
        l_quantity < 18.2811003923472 AND l_discount < 0.023341357999829684 AND
l_discount > 0.018050010074103496 AND l_shipdate < '1997-08-21'::date AND
l_shipdate > '1997-08-12'::date
        OR
        l_quantity < 21.035837899903893 AND l_discount < 0.02834339277229088 AND
l_discount > 0.024092944232228637 AND l_shipdate < '1997-11-19'::date AND
l_shipdate > '1997-03-12'::date
        OR
        l_quantity < 21.88697710685508 AND l_discount < 0.02689858944060175 AND
l_discount > 0.0038651513156768855 AND l_shipdate > '1997-02-11'::date
        OR
        l_quantity < 21.88697710685508 AND l_discount < 0.02689858944060175 AND
l_discount > 0.007071355868115796 AND l_shipdate > '1997-02-11'::date
        OR
        l_quantity < 21.88697710685508 AND l_discount < 0.02689858944060175 AND
l_discount > 0.012656307650684718 AND l_shipdate > '1997-02-11'::date
        OR
        l_quantity < 21.88697710685508 AND l_discount < 0.02840772987973926 AND
l_discount > 0.0016128480368708886 AND l_shipdate > '1997-01-18'::date
        OR
        l_quantity < 21.88697710685508 AND l_discount < 0.02840772987973926 AND
l_discount > 0.0016128480368708886 AND l_shipdate > '1997-02-04'::date
        OR
        l_quantity < 21.88697710685508 AND l_discount < 0.02840772987973926 AND
l_discount > 0.0016128480368708886 AND l_shipdate > '1997-02-11'::date
        OR
        l_quantity < 21.88697710685508 AND l_discount < 0.02840772987973926 AND
l_discount > 0.0038651513156768855 AND l_shipdate > '1997-02-11'::date
        OR
        l_quantity < 8.006048062149967 AND l_discount < 0.02840772987973926 AND
l_discount > 0.006111926262827921 AND l_shipdate < '1997-12-29'::date AND
l_shipdate > '1997-11-06'::date
        OR
        l_quantity < 9.020350886249453 AND l_discount < 0.017931561233652735 AND
l_discount > 0.007939723121103179 AND l_shipdate < '1997-11-04'::date AND
l_shipdate > '1997-10-26'::date
        OR
        l_quantity < 9.772429999364398 AND l_discount < 0.02636983897058536 AND
l_discount > 0.013967884488129835 AND l_shipdate < '1997-11-01'::date AND
l_shipdate > '1997-10-08'::date
        OR
        l_quantity < 11.342758059655893 AND l_discount < 0.02134765826100224 AND
l_discount > 0.018397866843956345 AND l_shipdate < '1997-08-13'::date AND
l_shipdate > '1997-05-08'::date
```

```
        OR
        l_quantity < 14.157215005900674 AND l_discount < 0.014810034508894808
AND l_discount > 0.007071355868115796 AND l_shipdate < '1997-09-02'::date AND
l_shipdate > '1997-07-13'::date
        OR
        l_quantity < 15.702815436753523 AND l_discount < 0.0245999444624751 AND
l_discount > 0.01676940262235773 AND l_shipdate < '1997-12-10'::date AND
l_shipdate > '1997-10-27'::date
        OR
        l_quantity < 18.26829224583729 AND l_discount < 0.0032273764692443366
AND l_discount > 0.0016128480368708886 AND l_shipdate < '1997-11-30'::date AND
l_shipdate > '1997-03-05'::date;

       CREATE INDEX c_104
        ON orders
        USING btree
        (o_orderdate);

       CREATE INDEX p_104
        ON orders
        USING btree
        (o_orderdate)
        WHERE o_orderdate < '1997-07-31'::date AND o_orderdate > '1997-01-26'::date
        OR
        o_orderdate < '1997-09-05'::date AND o_orderdate > '1997-02-06'::date
        OR
        o_orderdate < '1997-09-10'::date AND o_orderdate > '1997-03-12'::date
        OR
        o_orderdate < '1997-05-11'::date AND o_orderdate > '1997-01-07'::date
        OR
        o_orderdate < '1997-11-05'::date AND o_orderdate > '1997-01-24'::date
        OR
        o_orderdate < '1997-12-31'::date AND o_orderdate > '1997-11-23'::date
        OR
        o_orderdate < '1997-12-30'::date AND o_orderdate > '1997-06-15'::date
        OR
        o_orderdate < '1997-12-12'::date;


       CREATE INDEX p_303_306
        ON part
        USING btree
        (p_brand COLLATE pg_catalog."default", p_container COLLATE
pg_catalog."default")
        WHERE p_brand = 'BRAND#14 '::bpchar AND p_container = 'MED BAG'::bpchar
        OR
        p_brand = 'BRAND#14 '::bpchar AND p_container = 'MED BOX'::bpchar
        OR
        p_brand = 'BRAND#14 '::bpchar AND p_container = 'MED PACK'::bpchar
        OR
        p_brand = 'BRAND#14 '::bpchar AND p_container = 'MED PKG'::bpchar
        OR
        p_brand = 'BRAND#14 '::bpchar AND p_container = 'SM BOX'::bpchar
        OR
        p_brand = 'BRAND#14 '::bpchar AND p_container = 'SM CASE'::bpchar
        OR
        p_brand = 'BRAND#14 '::bpchar AND p_container = 'SM PACK'::bpchar
        OR
        p_brand = 'BRAND#14 '::bpchar AND p_container = 'SM PKG'::bpchar;
```