WENDEL GÓES PEDROZO

ARQUITETURA PARA SELEÇÃO DE ÍNDICES EM BANCO DE DADOS RELACIONAIS, UTILIZANDO ABORDAGEM BASEADA EM CUSTOS DO OTIMIZADOR

Dissertação apresentada como requisito à obtenção do grau de Mestre, Programa de Pós-graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientadora: Profª Drª Maria Salete M. G. Vaz.

CURITIBA

2008

WENDEL GÓES PEDROZO

ARQUITETURA PARA SELEÇÃO DE ÍNDICES EM BANCO DE DADOS RELACIONAIS, UTILIZANDO ABORDAGEM BASEADA EM CUSTOS DO OTIMIZADOR

Dissertação apresentada como requisito à obtenção do grau de Mestre, Programa de Pós-graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientadora: Profª Drª Maria Salete M. G. Vaz.

CURITIBA

2008

WENDEL GÓES PEDROZO

ARQUITETURA PARA SELEÇÃO DE ÍNDICES EM BANCO DE DADOS RELACIONAIS, UTILIZANDO ABORDAGEM BASEADA EM CUSTOS DO OTIMIZADOR

Dissertação aprovada como requisito parcial à obtenção do grau de Mestre no Programa de Pós-graduação em Informática da Universidade Federal do Paraná, pela Comissão formada pelos professores:

Orientadora: Profª Drª Maria Salete Marcon Gomes Vaz
Departamento de Informática, UFPR

Profª Drª Deborah Ribeiro Carvalho
Universidade TUIUTI do Paraná

Prof. Dr. Marcos Sfair Sunyé
Departamento de Informática, UFPR

Curitiba, 27 de Agosto de 2008

AGRADECIMENTOS

A Deus que me concedeu condições e forças para chegar até aqui, principalmente durante os meses de trabalho intenso durante o curso de mestrado.

A minha orientadora professora Maria Salete Marcon Gomes Vaz pela confiança depositada em mim, principalmente na fase de concepção deste trabalho, e pela motivação, opiniões e amizade durante a construção e finalização do trabalho.

Aos meus pais Benedito e Elma, pelo amor e apoio incondicional em todos os desafios e conquistas realizados até aqui. As minhas irmãs Cibele e Ligia, que mesmo a distância sempre me incentivaram e apoiaram nas atividades acadêmicas.

Ao Professor Rodolfo M. Barros, o qual tive a oportunidade de trabalhar na graduação (UNOPAR) e especialização (UEL). Seu apoio, cartas de recomendação, e nossos artigos publicados, foram de suma importância para ingresso no mestrado.

Aos professores do Departamento de Informática da UFPR, que me possibilitaram ter um crescimento acadêmico e profissional. Em especial aos professores da área de Banco de Dados / Tecnologia da Informação: Marcos Sfair Sunyé, Carmem Satie Hara e Laura Sanches Garcia os quais me transmitiram questionamentos, idéias e ensinamentos importantes para a qualidade desta dissertação. Ao professor Alexandre Ibrahim Direne pela dedicação na coordenação do curso de mestrado.

Agradeço aos amigos da Cinq Technologies, principalmente ao Aldir e Quiroga que sempre me apoiaram neste caminho, suprindo minha ausência quando precisei.

Aos demais amigos, colegas e funcionários do Departamento de Informática.

Um agradecimento especial a minha esposa Claudia, que esteve comigo desde o inicio do mestrado. Obrigado pela compreensão, amor, entusiasmo e auxílio durante o trabalho.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A seleção de índices corresponde a uma atividade, comumente, realizada por Administradores de Sistemas de Banco de Dados, para acelerar o desempenho de comandos submetidos a um SGBD relacional. Devido a complexidade dessa atividade, diversos trabalhos na literatura e em sistemas comerciais procuram produzir ferramentas que possam apoiar o administrador na escolha dos melhores índices, para uma determinada carga de trabalho. No entanto, grande parte dos trabalhos tem suas soluções, totalmente, separadas e com modelos de custos diferentes dos utilizados pelo Sistema Gerenciador de Banco de Dados.

Esta dissertação propõe uma arquitetura para solução do problema de seleção de índices em bancos de dados relacionais. Foi implementado um algoritmo, que analisa comandos SQL submetidos ao Sistema Gerenciador de Banco de Dados e recomenda os índices, para aumentar o desempenho dos comandos. Na arquitetura para seleção de índices foi utilizada, a abordagem baseada em custos do Otimizador, para solução do problema, bem como efetuado um estudo de caso, utilizando o Sistema Gerenciador de Banco de Dados PostgreSQL. Os benefícios e eficácia da arquitetura para seleção de índices foram atestados, utilizando como referencia a metodologia especificada pelo Benchmark TPC-C.

ABSTRACT

The index selection correspond to an activity, commonly carried through for Administrators of systems of database, for to speed up the performance of commands submitted to a relational DBMS. Due to complexity of this activity, diverse works in literature and in commercial systems seek produce tools that can assist the Administrator in the choice of the best indexes, for a given workload. However, great part of the works has its solutions totally separate and with models of costs different from those used by the Database Management System.

This thesis proposes an architecture for solving the problem of index selection in relationals databases. An algorithm was implemented that analyzes queries submitted to the Database Management System and recommends the indexes to increase the performance of the commands. In architecture for index selection was used, the approach based on costs of Otimizer for solution of the problem, and made a case study, using o Database Management System PostgreSQL. The benefits and effectiveness of the architecture for index selection were certified, using as reference the methodology specified for Benchmark TPC-C.

CAPÍTULO 1

INTRODUÇÃO

Esta dissertação apresenta um estudo de como prover mecanismos automáticos para a Seleção de Índices em sistemas de bancos de dados relacionais. A seguir, são especificados o problema e os objetivos do desenvolvimento desta pesquisa, bem como, a descrição de como a dissertação está organizada.

1.1 Motivação

Os atuais bancos de dados oferecem inúmeros parâmetros e configurações, os quais podem ser ajustados para alcançar um funcionamento mais eficiente. O ajuste desses parâmetros é complexo, pela quantidade existente, pela necessidade de compreensão dos algoritmos usados e pela possibilidade de inter-relações entre os ajustes de desempenho.

O profissional responsável pelos ajustes em um banco de dados, o DBA (*Database Administrator – Administrador de Banco de Dados*), normalmente desempenha duas atividades prioritárias: garantir a disponibilidade dos dados e assegurar níveis de desempenho aceitáveis. A primeira engloba tarefas, tais como: criação física de bancos de dados; geração de objetos (tabelas, índices, etc.); criação de políticas de cópias de segurança e recuperação, as quais garantem tempos mínimos de indisponibilidade dos dados; e proteção contra acessos indevidos.

A segunda atividade relaciona-se com o grau de satisfação dos usuários em relação ao tempo de resposta para comandos submetidos ao Sistema Gerenciador de Banco de Dados - SGBD e, normalmente, resulta da correta aplicação de medidas prévias de ajustes, tarefa conhecida por *Tunning* [34]. Dado ao permanente funcionamento do banco de dados, realizar essa tarefa demanda a maior parte das atividades dos esforços de um DBA.

Segundo [34], com os avanços no *hardware* e o desenvolvimento da WEB, os Sistemas de banco de dados teriam um crescimento acentuado nos próximos anos, bem como o volume de dados gerenciados por eles. Dessa forma, um crescente número de profissionais especializados, seria necessário para realizar o ajuste de desempenho das complexas aplicações de bancos de dados existentes. Como pode ser observado, à medida que os bancos de dados vão se tornando cada vez mais comuns, esta dependência de recursos humanos se torna indesejável.

Outro fator a ser considerado advém da constante evolução dos SGBDs, tais como Oracle [27], SQL Server [36], DB2 [16] e PostgreSQL [29]. Por exemplo, na Versão 7 do SGBD Oracle, havia necessidade de configuração manual para alocação de espaços em disco para os segmentos (tabelas, índices etc.), bem como, acompanhar o crescimento e tomar medidas corretivas quando da constatação de níveis críticos de fragmentação. Esse procedimento deixou de ser necessário nas versões subseqüentes, já que as tarefas de alocação e manutenção de espaço foram automatizadas e a necessidade de intervenções manuais por parte dos DBAs vem sendo reduzida.

A dependência de mão de obra especializada, em oposição às constantes melhorias em ferramentas, têm gerado demanda de ferramentas de auxílio aos DBAs. Alguns trabalhos [1,15,41] contribuem com métodos/ferramentas para auxiliar o DBA nas atividades relativas à correção de desempenho.

Em um SGBD Relacional as perdas de desempenho são oriundas de diversas causas, e podem ser devido a fatores externos e internos. Como fatores externos temos o processador, a memória, os discos, assuntos relacionados a conectividade e o sistema operacional. Já os fatores internos, que contribuem para a perda de desempenho, podem ser os níveis de concorrência, a configuração de parâmetros e a configuração de índices.

Em [14] é citado inúmeros casos onde o trabalho visando reduzir a carga computacional de um Comando SQL *(Structured Query Language)* pode levar a ganhos significativos através de

ajustes de índices, ganhos esses dificilmente atingidos usando outros recursos, como alterar os parâmetros globais do SGBD, realizar alterações no Sistema Operacional, entre outros.

Apesar da importância da seleção e ajuste adequados de índices para uma carga de trabalho transacional, essa tarefa pode ser complexa, principalmente se forem consideradas as aplicações OLTP (*Online Transaction Processing* - Processamento de transações em tempo-real) de médio e grande porte, as quais podem ter milhares de tabelas, onde cada tabela pode ter centenas de colunas, resultando em um espaço de trabalho de milhões de consultas. Neste cenário, o número de possíveis índices a considerar é enorme e encontrar o conjunto de índices que otimize esse espaço de trabalho, é um desafio combinatório e um problema científico aberto a pesquisa.

Outro ponto a ser discutido, é sobre as ferramentas de auxilio ao DBA para seleção de índices. Grande parte das ferramentas existentes é construída totalmente separada do SGBD e independentemente propõem índices candidatos e tentam avaliar os custos e benefícios de cada conjunto de índice, quando submetido ao SGBD. Isto não é desejável, uma vez que o SGBD pode não realizar as mesmas suposições sobre custos da ferramenta, produzindo uma discrepância entre o modelo de custo da ferramenta e o modelo de custos utilizado pelo SGBD.

Esta dissertação se concentra na pesquisa de uma solução para automatização da tarefa de seleção de índices para instruções SQL, visando obter o melhor desempenho possível para consultas submetidas a um SGBD. sem realizar nenhuma melhoria nos recursos de hardware disponíveis.

1.2 Objetivos

O objetivo principal deste trabalho é apresentar a construção de uma arquitetura para seleção de índices em banco de dados relacionais. A arquitetura é responsável pela análise dos comandos SQL submetidos ao SGBD e pela recomendação de índices, visando aumentar o desempenho dos comandos.

Outro objetivo do trabalho foi criar uma heurística para seleção de índices candidatos. Essa heurística é responsável pela análise dos predicados e das cláusulas presentes nos comandos SQL submetidos e encontrar colunas que, possivelmente, poderiam trazer benefícios de desempenho ao comando.

1.3 Estrutura da Dissertação

O restante desta dissertação está organizado como segue. O Capítulo 2 apresenta uma revisão bibliográfica sobre índices e seleção de índices em Bancos de Dados Relacionais. O Capítulo 3 descreve como é o modelo de arquitetura para seleção de índices proposto, bem como a heurística desenvolvida. No capítulo 4 é feito um estudo de caso utilizando o SGBD PostgreSQL e discutidas as questões de implementação da arquitetura para seleção de índices no SGBD PostgreSQL. O Capítulo 5 traz uma avaliação de desempenho da arquitetura utilizando uma carga de trabalho OLTP, bem como os resultados e uma comparação com trabalhos relacionados. Finalmente, o Capítulo 6 apresenta as conclusões e as perspectivas de trabalhos futuros.

CAPÍTULO 2

ÍNDICES EM BANCOS DE DADOS RELACIONAIS

Antes de abordar o problema tratado por esta dissertação, cabe realizar uma revisão de conceitos relativos à criação de índices em sistemas de bancos de dados relacionais, seleção de índices e sobre técnicas utilizadas para solução dos problemas de seleção de índices.

2.1 Introdução

Um índice é uma organização de dados que permite acelerar o tempo de acesso as linhas de uma tabela. Uma abordagem semelhante ao conceito de índices em banco de dados é o índice remissivo, utilizado em livros, onde os termos e os conceitos procurados freqüentemente pelos leitores são reunidos em um índice alfabético, colocado ao final dele. O leitor pode percorrer o índice rapidamente e ir para a página desejada. Assim como, é tarefa do autor prever os itens que os leitores provavelmente vão procurar. É tarefa do DBA prever quais índices trarão benefícios.

Segundo [19], em Bancos de dados, existem três razões para a definição de índices. Uma é permitir que as linhas sejam acessadas rapidamente através do valor do atributo indexado. A segunda é facilitar a ordenação das linhas por aquele atributo. Já a terceira razão é concernente à unicidade, quando é necessário que uma coluna seja única, um índice é criado pelo SGBD com a função de assegurar que não sejam aceitos valores duplicados.

Para exemplificar melhor os benefícios da utilização de índices, considere o exemplo da Figura 2.1, sobre a tabela Clientes.

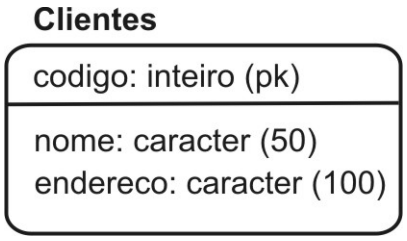

Figura 2.1: Tabela Clientes

Considere também uma consulta SQL sobre a tabela (Figura 2.1):

```
SELECT nome FROM clientes WHERE codigo = 2;
```

Sem índices, o sistema precisaria varrer toda a tabela `clientes`, linha por linha, para encontrar todas as entradas correspondentes. Havendo muitas linhas em `clientes`, e poucas ou nenhuma linha retornadas pela consulta. Esse método é claramente ineficiente. Porém, se o sistema fosse instruído para manter um índice para a coluna `codigo`, então poderia ser utilizado um método mais eficiente para localizar as linhas correspondentes.

Em SGBDs Relacionais, existem dois tipos básicos de índices: índices ordenados e índices *hash* [19]. O primeiro baseia-se em criar uma estrutura de índice para cada chave de pesquisa. Os valores das chaves são armazenados de forma ordenada, possibilitando assim, acesso rápido aos registros associados a cada chave de pesquisa.

Os índices *hash* baseiam-se em distribuir uniformemente os valores de chaves para uma determinada faixa no disco de armazenamento, denominada *bucket.* Para se obter um registro, é aplicada uma função, denominada função *hash* sobre a chave de pesquisa, e então é obtido o *bucket* que contém aquele registro [19].

2.2 Estrutura de Dados para Implementação de Índices

Uma chave de um índice é uma seqüência de atributos, ligados aos registros correspondentes. Em sistemas relacionais, a estrutura mais comumente utilizada para representar índices são as árvores.

Uma árvore é uma estrutura de dados cujos elementos têm relacionamentos um para muitos entre si. Cada elemento tem no máximo um pai. De acordo com a terminologia padrão, cada elemento da árvore é chamado de nó, e os relacionamentos entre elementos são chamados ramos. A profundidade de um nó é a distância desse até a raiz. Um conjunto de nós, com mesma profundidade, é denominado nível da árvore e o número máximo de descendentes para cada um dos nós é denominado grau da árvore. A maior profundidade de um nó é a altura da árvore [19].

Uma árvore pode ser balanceada ou degenerada. Uma árvore é dita balanceada quando suas sub-árvores à esquerda e à direita possuem a mesma altura [40]. Quando uma árvore não atende a esse quesito, esta é considerada uma árvore degenerada.

De acordo com sua estrutura de distribuição uma árvore pode ser classificada em vários tipos: binária, AVL (arvore balanceada pela altura), ordenada, B, B+[39]. Neste trabalho, foi restringido o uso de índices a árvores B+, devido a restrição do SGBD PostgreSQL utilizado no estudo de caso, em somente permitir a criação de índices em múltiplas colunas, utilizando árvores B+..

Uma Árvore B+ é uma árvore balanceada, cujas folhas contêm uma seqüência de pares chave-ponteiro. As chaves são ordenadas pelo seu valor [8]. Um exemplo deste tipo de estrutura é mostrado na Figura 2.2.

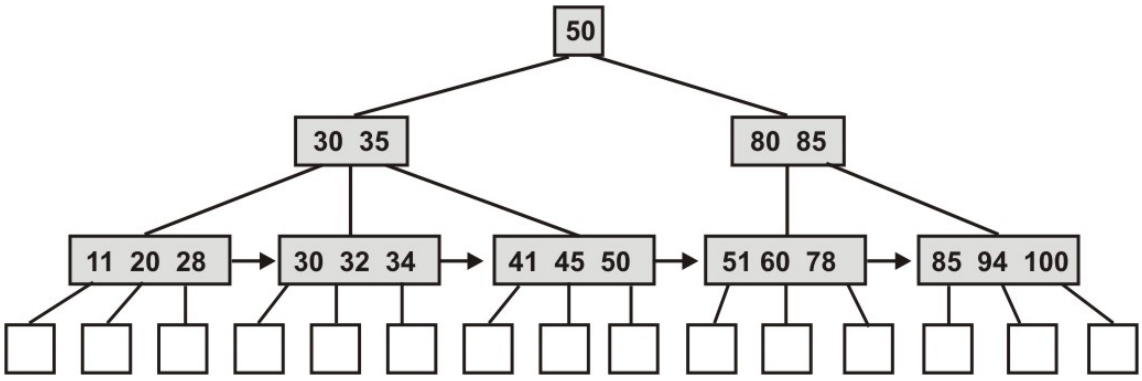

Figura 2.2 - Exemplo de Árvore B+

Uma característica que torna as Árvores B+ adequadas para o uso em sistemas de banco de dados é o armazenamento de diversas chaves por nó, criando uma árvore de grau alto. Isso implica que a altura da árvore tende a ser pequena, mesmo para um conjunto considerável de chaves. Os ganhos de desempenho para o SGBD é devido principalmente a possibilidade de indexar milhões de chaves mantendo a altura 3 (três) da árvore [21]. Como a obtenção de cada nó é feita por um acesso ao disco, pode-se consultar informações com uma determinada chave, após poucos acessos.

Quando é necessário fazer inserções ou remoções de chaves na árvore, existem algoritmos apropriados que fazem reestruturações locais, garantindo assim, a propriedade de balanceamento da árvore.

A quantidade de reestruturações feitas por esses algoritmos, durante uma atualização, depende do quanto cada nó da árvore está preenchido em relação à sua capacidade total. Em alguns sistemas de banco de dados, como o Oracle [27] e o Microsoft SQL Server [36], o DBA pode especificar um parâmetro de fator de preenchimento para as páginas dos índices, representado como Árvores B+. Esse fator de preenchimento regula o quanto cada nó da árvore pode ficar preenchido, facilitando ou dificultando as atualizações.

As folhas da árvore são ordenadas pela chave do índice e ligadas por ponteiros. Isso permite que consultas que acessam um intervalo de valores da chave sejam processadas eficientemente. O acesso às tuplas da relação é feito seguindo os ponteiros associados a cada chave no nível folha.

2.3 Índices Ordenados

Os índices ordenados são utilizados para se obter acesso aleatório, eficiente, aos registros de um arquivo. Esses índices armazenam os valores das chaves de pesquisa, de forma ordenada, associando cada chave de pesquisa aos registros que contém aquela chave.

Assim, como os registros de um índice remissivo de livros são impressos em ordem

alfabética, os registros em um arquivo indexado podem estar armazenados seguindo alguma ordem. Baseados nessa ordem, os índices ordenados podem ser classificados em primários e secundários.

2.3.1 Índices Primários e Secundários

Quando o arquivo que contém os registros, está ordenado seqüencialmente, o índice associado a esse arquivo é chamado de índice Primário ou *clustering* [18]. Geralmente, a chave de pesquisa de um índice primário é a própria chave primária [18]. Devido a essas características, consultas por períodos que acessam informações da tabela, produzem melhores resultados quando realizadas utilizando um índice primário.

Um índice é definido como secundário, quando as chaves de pesquisa especificam uma ordem diferente da ordem seqüencial do arquivo indexado, ou seja, a ordem das chaves nas folhas do índice é diferente das linhas armazenadas na tabela [18].

2.3.2 Índices Densos e Esparsos

Os ponteiros das chaves, no nível folha, da árvore podem apontar tanto para os registros da tabela subjacente quanto para os blocos em que estes registros estão armazenados. Um índice é denominado denso, quando um registro do índice aparece para cada registro do arquivo.

Nesse tipo de implementação o registro do índice contém o valor da chave de pesquisa e um ponteiro para o primeiro registro de dados com o valor da chave de pesquisa [18], conforme ilustrado na Figura 2.3.

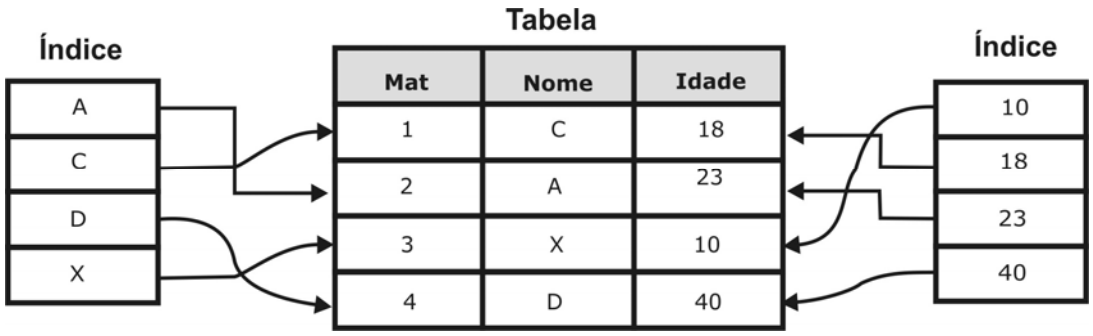

Figura 2.3: Exemplo de Índices Densos

O índice esparso corresponde a um registro de índice, criado apenas para alguns dos valores. Assim como nos índices densos, cada registro do índice contém um valor de chave de pesquisa e um ponteiro para o primeiro registro de dados com esse valor de chave de pesquisa [18], conforme pode ser visto na Figura 2.4.

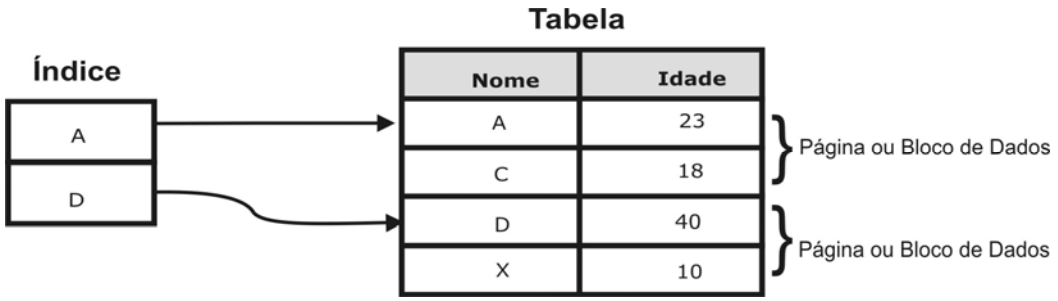

Figura 2.4: Exemplo de Índices Esparsos

2.3.3 Índices Completos e Parciais

Os Índices completos são aqueles que indexam todas as linhas de uma determinada tabela. Os índices parciais indexam somente um subconjunto de linhas definidas por um predicado na criação do índice.

Índices parciais são úteis quando há necessidade de indexação de colunas que tenham as distribuições de dados não uniformes, pois exclui a representação de valores com grandes quantidades de repetições.

2.4 Índices com Múltiplas Colunas

O índice em múltiplas colunas pode ser definido contendo mais de uma coluna. Para melhor compreensão, considere a tabela `Funcionarios` na Figura 2.5.

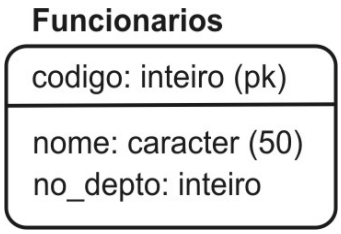

Figura 2.5: Tabela Funcionarios

Supondo que freqüentemente sejam feitas consultas SQL como:

```
SELECT nome FROM funcionarios WHERE codigo = 2 AND no_depto = 15;
```

É apropriado definir um índice contendo as colunas `codigo e no_depto`. Em um SGBD, o Otimizador, módulo responsável por encontrar a melhor forma de recuperar os dados, pode utilizar um índice com várias colunas, para comandos envolvendo a coluna mais a esquerda na definição do índice, e quaisquer colunas listadas a sua direita, sem omissões. Por exemplo, um índice contendo as colunas `codigo, nome e no_depto`, pode ser utilizado em comandos, envolvendo `codigo, nome e no_depto`, ou em comandos com `codigo e nome`, ou em comandos com apenas `codigo`, mas não em outras combinações. Em um comando envolvendo `codigo e no_depto`, o Otimizador pode utilizar o índice apenas para `codigo`, tratando `no_depto` como uma coluna comum não indexada.

Os índices com várias colunas só podem ser utilizados se as cláusulas envolvendo as colunas indexadas forem ligadas pelo operador lógico de conjunção “AND”, utilizado sempre que for necessário todas as condições estabelecidas verdadeiras.

2.5 Seleção de Índices

Apesar de grande esforço dos DBAs, existem sérios problemas de desempenho em SGBDs causados por configurações incorretas de índices. Ainda, há muito desconhecimentos sobre a necessidade de criá-los ou não. Para melhor visualizar esses problemas, considere o exemplo a seguir, onde 2 tipos de comandos SQL são submetidos, sobre uma tabela denominada `Funcionarios`:

```
(1)  insert into Funcionarios (codigo, nome, salario, no_depto)
     values (4, 'Fulano da silva', 5000, 7);

(2)  select codigo, nome, salario
     From Funcionarios
     where salario between  2500   and  5000
     order by nome;
```

Supondo, inicialmente, que os comandos são submetidos ao SGBD de forma concorrente e com uma freqüência de comandos do Tipo 2, muito maior do que os comandos do Tipo 1. Nesse cenário, um índice sobre as colunas `salario` e `nome` da tabela `Funcionarios` pode trazer benefícios de desempenho no processamento das transações onde estão inseridos.

Os comandos do Tipo (1) são menos freqüentes, e assim é esperado um custo menor, decorrente da atualização do índice sobre a tabela `Funcionarios`, compensando a criação do mesmo. De outra forma, se tivermos a freqüência de comandos do Tipo (1) muito superior do que a freqüência de comandos do Tipo (2), a criação de índices não é recomendável, devido ao custo de atualização dos índices presentes sobre a tabela.

Tradicionalmente, os índices podem aumentar a velocidade de execução de consultas em SGBDs relacionais nos seguintes casos:

- Limitar dados a ser acessados, quando se aplicam predicados;

- Ordenações de registros utilizando cláusulas como ORDER BY ou GROUP BY;
- Fazer junção de uma tabela com outra(s); e
- Eliminar registros repetidos, utilizando a Cláusula DISTINCT.

Para automatizar essa atividade em SGBDs relacionais, muitos pesquisadores [5, 10, 11, 23] estão trabalhando no intuito de buscar soluções/metodologias para a seleção de índices.

2.6 O problema de Seleção de Índices

Na teoria de algoritmos, problemas tratáveis, ou seja, solúvel por algum algoritmo onde o número de computações cresce polinomialmente em função do tamanho da instância, pertencem à classe P (*Polinomial Time*), de outra forma, os problemas intratáveis pertencem à classe NP (*Nondeterministic Polinomial Time*) [35].

Atualmente, a classe de Problemas NP é onde reside a maioria dos problemas computacionais, por conter os problemas difíceis de se tratar e pelo fato do número de computações do melhor algoritmo conhecido, crescer exponencialmente em função do tamanho da instância.

Já foi demonstrado em [7], que dependendo do tamanho da instância, o problema de seleção de índices se enquadra em uma sub-classe dos Problemas NP, denominado NP-Dificil. De fato, encontrar o conjunto ótimo de índice adequado a uma carga de trabalho não é uma atividade trivial. Para exemplificar esse problema, considere uma tabela com *n* colunas, a qual o interesse é descobrir o número de diferentes índices possíveis com *k* colunas, onde $k <= n$. Para a primeira coluna, existem *n* chances, já a segunda tem *n-1* chances restantes. Para as demais colunas adicionadas, temos como total $n(n-1)(n-2)... (n-k+1)$ ou $n! / (n-k)!$. Segundo [31], o número total de diferentes índices que é possível sobre uma tabela com *n* colunas pode ser conhecido utilizando a seguinte fórmula:

$$\sum_{k=1}^{n} \frac{n!}{(n-k)!}$$

Para melhor compreensão da formula, considere o seguinte exemplo sobre uma tabela com 10 colunas a serem indexadas. Aplicando a fórmula para *n = 10* e *k = 1, k = 2 e k =5* respectivamente, temos os seguintes resultados:

- 10 índices diferentes de 1 coluna;
- 90 índices diferentes de 2 colunas; e
- 30240 índices diferentes de 5 colunas.

Portanto, para um ambiente envolvendo centenas de tabelas, como acontece nos ambientes OLTPs atuais, encontrar o conjunto ótimo de índices adequado, é um grande desafio combinatório.

2.7 Metodologia para Seleção de Índices

Alguns trabalhos [5,10,11,23] tratam o problema de seleção de índices, construindo ferramentas de apoio ao DBA, na escolha de índices para uma determinada carga de trabalho $W=\{(Q_i,f_i), i=1,...,n\}$, *onde* Q_i é uma consulta SQL e f_i é a sua freqüência de submissão.

As ferramentas implementam algoritmos que objetivam minimizar o custo total de processamento de *W* pelo sistema. O custo total de processamento é a soma dos custos de acesso, atualização e manutenção dos índices.

Esses trabalhos apresentam uma metodologia comum para a escolha de índices que são adequados a uma carga de trabalho (Figura 2.6).

Figura 2.6: Processo de seleção de índices

O primeiro passo para a seleção de índices é obter a Carga de Trabalho sobre a qual a ferramenta operará. Normalmente, a ferramenta possui duas formas de aquisição dos comandos SQL e suas freqüências de execução. A primeira forma é manual, onde o DBA insere uma lista de comandos SQL e suas freqüências estimadas.

A segunda forma é feita através de arquivos de *trace*. O DBA usa um utilitário do sistema que permite o registro, em um arquivo de estatística de todos os comandos SQL, submetidos pelos usuários. Esse arquivo é processado para detectar quantas vezes os comandos SQL foram executados durante o período de coleta.

Em alguns sistemas [16,27], é possível obter os comandos recentemente executados através da varredura da *cache*, reservada para operações SQL. Em seguida, para cada comando da carga de trabalho, é aplicada uma heurística de escolha de índices candidatos, que irá determinar os melhores índices para cada comando SQL. Existem diversas abordagens para a construção de heurística de escolha de índices candidatos. Existem estratégias baseadas em regras [6,32] e as estratégias baseadas em custos do Otimizador [4,10,23].

Na primeira abordagem, a escolha de quais índices, para um comando SQL, é baseada no uso de regras derivadas a partir do conhecimento de especialistas. Um exemplo de regra, seria sempre criar um índice para consultar uma tabela com mais de 50 blocos e filtrada por um predicado que obtêm menos de 5% das tuplas.

Na segunda abordagem, o próprio Otimizador de consultas é utilizado para classificar quais índices podem gerar maior benefício na execução do comando. E como é mostrado na seção 2.8, esta abordagem é relacionada às técnicas baseadas em custos do Otimizador.

Finalmente, após a escolha dos índices candidatos, é aplicada uma heurística final de seleção de índices para determinar quais índices oferecem a melhor relação custo-benefício para a carga de trabalho como um todo. Esse tipo de heurística pode considerar o compromisso existente entre os custos de manutenção dos índices e as melhorias trazidas por estes na execução de consultas.

Após a aplicação da heurística final de seleção de índices, as ferramentas recomendam criações ou remoções de índices pelo DBA, que irá decidir se, e quando, estas recomendações deverão ser efetivadas.

2.8 Seleção de Índices Baseados em Custos do Otimizador

Um grande avanço no projeto de ferramentas de auxílio ao DBA, foi o uso do Otimizador do SGBDs para avaliar os custos de criação e manutenção dos índices, dado um conjunto de índices candidatos [10,23].

A utilização do modelo de custos do Otimizador do SGBD, para auxiliar no problema de seleção de índices, teve seu início com os trabalhos de [10] e [11]. Um ponto a ser destacado na utilização desta técnica é que não é necessário desenvolver modelos de custos separados do SGBD**.**

A utilização do Otimizador no processo de seleção de índices propõe que o esquema do banco de dados seja povoado com configurações hipotéticas, antes do processo iniciar. Uma

configuração hipotética representa estruturas presentes no banco de dados, mas não materializada fisicamente. No entanto, essas estruturas podem ser reconhecidas como válidas pelo SGBD, para estimativas de custos e planos de acesso, tais como, índices e tabelas.

Para utilização de configurações hipotéticas, é necessário estender o SGBD para possibilitar a definição de estruturas físicas hipotéticas. É preciso que o SGBD dê suporte a algum mecanismo, que possibilite simular a presença de um índice ou tabela na base de dados, sem que seja necessário materializar, fisicamente, esses objetos.

É necessário, também, possibilitar que o Otimizador do sistema gere planos de execução e estimativas de custos, considerando a existência de estruturas hipotéticas. Esta última propriedade permite utilizar o modelo de custos existente no Otimizador, evitando a criação de um modelo de custos especializado na ferramenta de seleção de índices.

Para configurações hipotéticas são utilizadas técnicas tais como: criação de réplicas e índices hipotéticos.

2.8.1 Criação de Réplicas

O primeiro trabalho utilizando Réplicas foi de Finkelstein et al [10]. Essa abordagem considera que na seleção de índices para um determinado banco de dados, sejam criadas réplicas, no catálogo do SGBD, das tabelas envolvidas. Essas réplicas não possuem extensão física, mas recebem todas as informações estatísticas das tabelas originais.

Para simular uma configuração de índices, são criados índices reais sobre as réplicas (tabelas vazias) e as estatísticas destes são corrigidas pela ferramenta de seleção de índices.

Para obter estimativas de custos de execução de uma consulta sob uma configuração hipotética, a ferramenta de seleção de índices troca as referências a tabelas na consulta, pelas suas respectivas réplicas, as quais estão povoadas com índices hipotéticos.

2.8.2 Índices Hipotéticos

Outra abordagem para utilização de configurações hipotéticas aconselha que o esquema do SGBD seja temporariamente populado, com índices hipotéticos. Essa abordagem foi utilizada no SGBD IBM DB2 UDB [16] e também utilizada neste trabalho. Na Figura 2.7, as etapas para a seleção de índices utilizando índices hipotéticos são apresentadas.

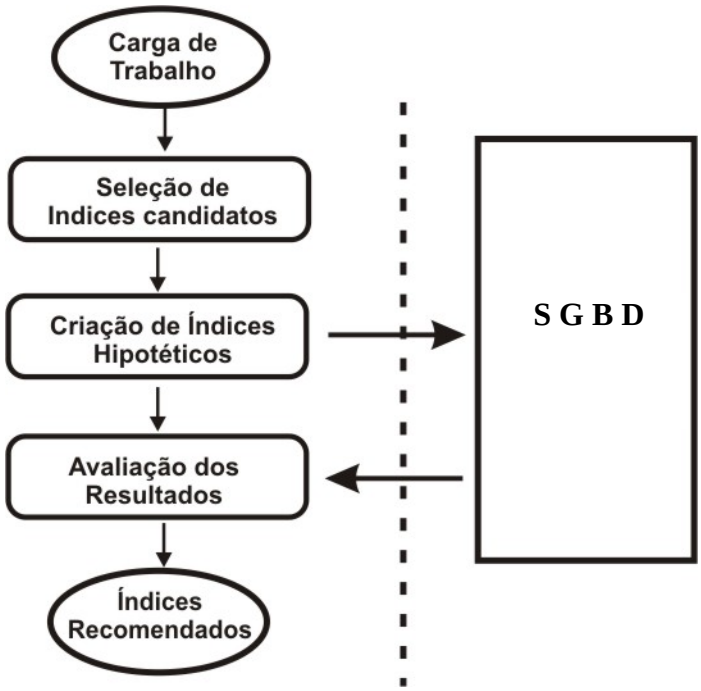

Figura 2.7: Etapas para seleção de índices

Conforme Figura 2.7, a partir da carga de trabalho, contendo comandos SQL, é utilizada uma heurística para enumerar índices candidatos para um comando SQL e inserir esses no esquema do SGBD. Em seguida, é iniciado o processo de otimização do comando, onde o Otimizador faz o plano de execução, que pode conter um ou vários índices hipotéticos, constantes no catálogo do SGBD. Após o Otimizador gerar o plano de execução, é feita uma avaliação com o propósito de enumerar os índices hipotéticos constantes no plano gerado, recomendando-os ao DBA.

As vantagens dessa abordagem, em relação à criação de réplicas, são: (i) com índices hipotéticos é possível fazer a simulação de configurações hipotéticas de forma mais simples e eficiente; e (ii) na criação de réplicas é necessário que uma ferramenta externa tenha acesso as informações críticas do catálogo do SGBD.

CAPÍTULO 3

ARQUITETURA PARA SELEÇÃO DE ÍNDICES BASEADA EM CUSTOS DO OTIMIZADOR

Neste capítulo, é apresentada a arquitetura para seleção de índices baseada em custos do Otimizador resultado desta dissertação. O objetivo é descrever formalmente todos os componentes envolvidos na arquitetura, mas sem vincular a nenhum SGBD específico. No capitulo 4 é utilizado um SGBD para estudo de caso do trabalho.

Inicialmente, na Seção 3.1 é apresentada uma visão geral da arquitetura, com a integração dos módulos e o fluxo das informações. A Seção 3.2 apresenta uma descrição da Heurística para escolha de índices candidatos e a sua importância no processo de seleção de índices. Na Seção 3.3 são descritos os cálculos de custos e benefícios dos índices sugeridos, bem como, um detalhamento das equações propostas para as estimativas. Por fim, na Seção 3.4 é apresentado o algoritmo desenvolvido, suas características e o seu papel dentro do processo de seleção de índices.

3.1 Visão Geral da Arquitetura

A arquitetura para seleção de índices visa resolver problemas de seleção de índices em bancos de dados relacionais. Embora exista grande esforço por parte dos DBAs o problema de seleção de índices é complexo e traz sérios problemas de desempenho a um SGBD, quando mal configurado.

Na Figura 3.1 pode-se visualizar a arquitetura, com a comunicação e integração entre todos os atores e módulos do sistema.

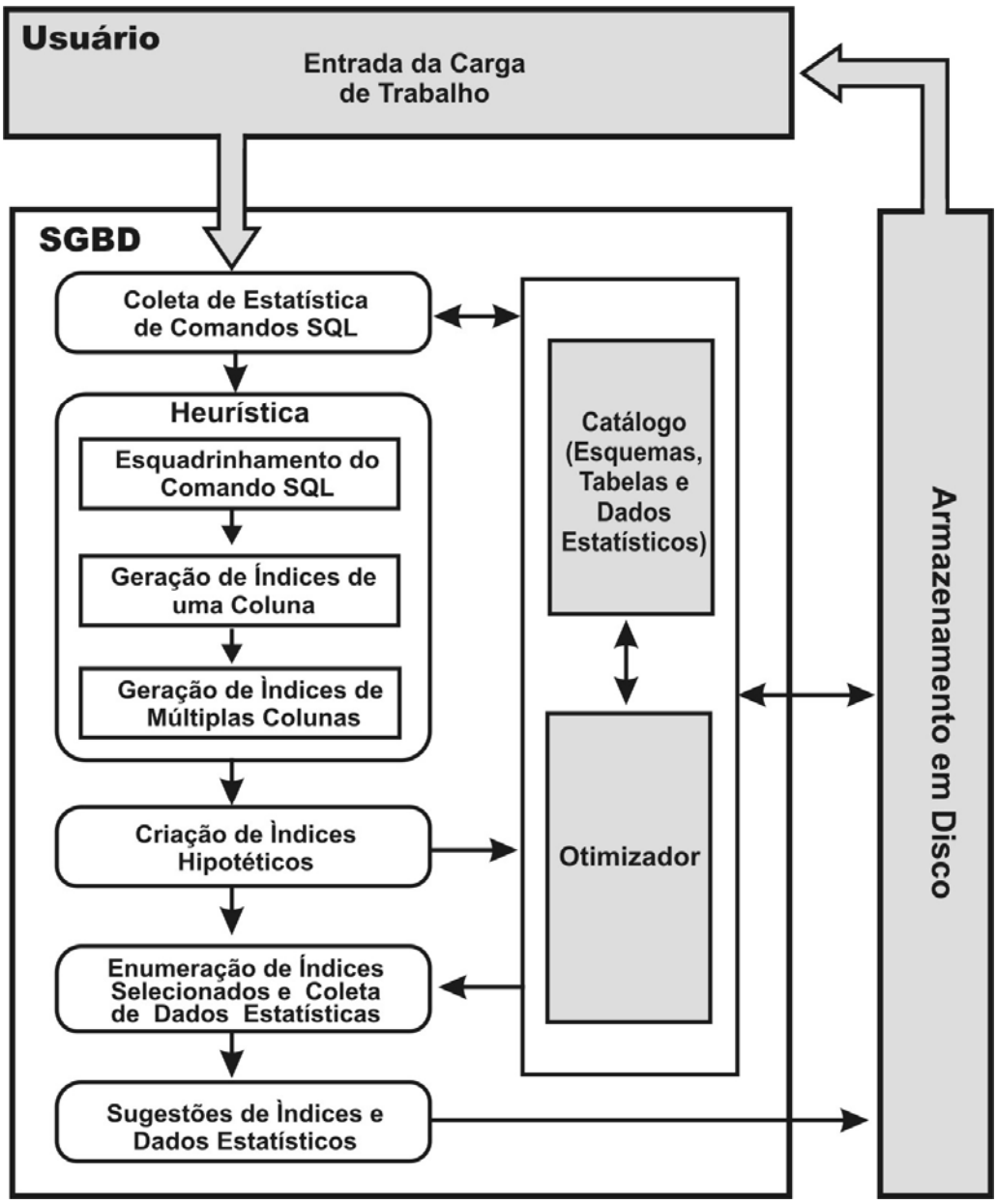

Figura 3.1: Arquitetura para seleção de índices

Na arquitetura proposta (Figura 3.1), o processo de seleção de índices, engloba seis etapas, as quais são a seguir:

(1) Carga de trabalho. Como primeira etapa, temos a entrada da carga de trabalho, contendo comandos SQL, inserida pelo usuário através interface de entrada e saída de dados no SGBD.

(2) Coleta de Estatísticas de Comandos SQL. Nesta etapa é solicitado ao Otimizador, dados estatísticos sobre os comandos SQL submetidos. Essas estatísticas são as mesmas que podem ser obtidas através do comando *Explain* do SGBD e refletem as configurações atuais da base de dados, antes do processo de seleção de índices iniciar.

(3) Heurística. Nesta etapa é realizada a seleção de índices candidatos, efetuada através do esquadrinhamento do comando, a fim de encontrar colunas que potencialmente podem trazer benefícios ao comando. Também são verificados e eliminados os índices candidatos gerados, que são idênticos aos que existem fisicamente no esquema.

(4) Criação de Índices Hipotéticos. Nesta etapa é efetuada a criação dos índices candidatos no catálogo do SGBD. Apesar de não estarem materializados fisicamente, os dados estatísticos dos índices são simulados, atribuindo aos mesmos, dados estatísticos da tabela vinculada a ele.

(5) Processo de Otimização. Com índices hipotéticos criados no esquema do banco de dados, o processo de otimização é efetuado normalmente, considerando os índices reais e hipotéticos durante o processamento do comando.

(6) Enumeração de Índices Selecionados e Coleta de Dados Estatísticos. Para conhecer quais foram os índices escolhidos pelo Otimizador durante o processamento, nesta etapa, é realizado o processo de enumeração dos índices, constantes no plano de execução gerado pelo Otimizador, a fim de, identificar quais destes são índices hipotéticos. Outra ação efetuada nesta etapa é a coleta de dados estatísticos, dos índices hipotéticos, encontrados no plano de execução.

(7) Sugestões de Índices e Dados Estatísticos. A última etapa no processo de seleção de índices é gerar sugestões de índices e dados estatísticos de custos e benefícios dos índices

recomendados. Esses dados são gravados em arquivo no disco e disponibilizados ao usuário. Outra atribuição desse módulo é alterar o catálogo do SGBD, no intuito de apagar todos os índices hipotéticos criados.

Na arquitetura para seleção de índices destacar-se que o próprio Otimizador faz a enumeração dos melhores índices para uma dada consulta, sem precisar materializar fisicamente os índices candidatos no banco de dados. Além de permitir uma simulação de configuração eficiente, a abordagem ainda garante que os índices recomendados são considerados pelos SGBDs, quando materializados fisicamente.

3.2 Heurística para Seleção de Índices Candidatos

O processo de recomendação de índices se dá em poucas etapas (Figura 3.1). Contudo, a etapa de seleção de índices candidatos é uma das etapas mais importantes no processo, pois os dados resultantes desta etapa são submetidos ao Otimizador do SGBD, para produção do plano de execução para o comando SQL.

Na literatura existem várias abordagens [5,10,11,23] para construção de heurísticas para seleção de índices candidatos. A Heurística, utilizada neste trabalho, foi desenvolvida com o objetivo de encontrar usos de colunas que poderiam ser indexadas. A Heurística procura encontrar no comando SQL, cinco tipos de usos de colunas (Figura 3.2)

1. Colunas envolvidas em predicados de igualdade;
2. Colunas envolvidas em cláusulas ORDER BY, GROUP BY e predicados de junção;
3. Colunas que aparecem em restrições de intervalos;
4. Colunas que aparecem em outros predicados indexados (por exemplo, *like*); e
5. Demais colunas referenciadas no comando SQL.

Figura 3.2: Usos de colunas para implementação de índices

O processo para a escolha dos índices candidatos se dá em 5 etapas (Figura 3.3). Na primeira etapa é feito uma varredura no Comando SQL, procurando índices candidatos de uma coluna, que satisfaçam os usos descritos. Na segunda etapa é feita uma combinação de todos os índices candidatos, de uma coluna, para formar índices candidatos de múltiplas colunas. Por fim, temos os candidatos finais, correspondentes a união dos índices candidatos de uma coluna, com os índices candidatos de múltiplas colunas.

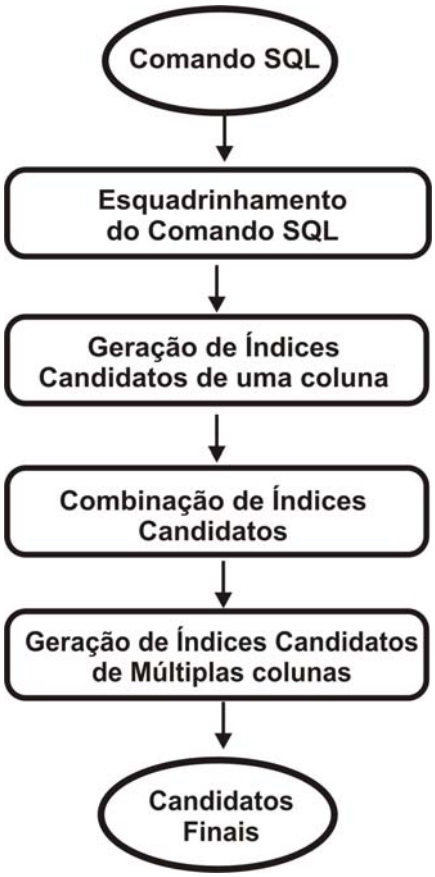

Figura 3.3: Fluxo para escolha de índices candidatos.

Para melhor exemplificar o fluxo produzido pela heurística de escolha de índices candidatos, considere a seguir, um estudo de caso utilizando a Tabela `Funcionarios`:

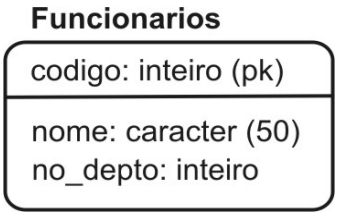

Figura 3.4: Tabela Funcionarios

Considerando, que no início do fluxo, seja submetido o seguinte comando SQL:

```
SELECT nome FROM Funcionarios
WHERE codigo = 10 AND
nome = "Fulano da Silva" AND no_depto = 5;
```

Na primeira etapa será feito o esquadrinhamento da consulta e selecionados três índices candidatos. Para escolha das colunas, a serem criados os índices, foi seguida as diretivas citadas na Figura 3.2.

Os dados dos índices candidatos gerados, consta na Tabela 3.1. No cabeçalho "Nº coluna" da Tabela 3.1, o nº 1 corresponde a coluna `codigo` da tabela `Funcionarios`, o nº 2 a coluna `nome`, e o nº 3 corresponde a coluna `no_depto`.

Tabela 3.1: Índices Candidatos para Tabela Funcionarios

Nome	Nº Coluna
indice_1	1
indice_2	2
indice_3	3

Na segunda etapa, é realizada uma combinação de todos os índices candidatos de uma coluna, selecionados anteriormente, para formar índices de múltiplas colunas. Segue na Tabela 3.2, os índices resultantes dessa combinação:

Tabela 3.2: Combinações de Índices para a Tabela Funcionarios

Nome	Nº coluna
indice_4	1,2
indice_5	1,3
indice_6	2,1
indice_7	2,3
indice_8	3,1
indice_9	3,2
indice_10	1,2,3
indice_11	1,3,2
indice_12	2,1,3
indice_13	2,3,1
indice_14	3,1,2
indice_15	3,2,1

Os índices finais gerados pela heurística são todos os índices gerados na Tabela 3.1, mais os índices gerados na Tabela 3.2.

A heurística utilizada tem como principal vantagem a enumeração exaustiva de todas as possibilidades de índices candidatos, que potencialmente podem trazer benefícios para o comando. Embora seja utilizada uma enumeração exaustiva para escolha de candidatos, a heurística utiliza para isso, somente colunas que aparecem no Comando SQL, e que se enquadram nos itens citados na figura 3.2, sobre usos de colunas a serem indexadas.

3.3 Cálculo de Custos e Benefícios de Índices Hipotéticos

A arquitetura para seleção de índices desenvolvida tem por objetivo dar apoio a um profissional especializado em banco de dados, através da sugestão de um conjunto de índices adequados a uma carga de trabalho dada como entrada. No entanto, é responsabilidade do DBA a decisão de criar ou não os índices recomendados. Para prover informações que auxiliem a tomada de decisão, são disponibilizadas informações estatísticas de custos de processamento do comando e também métricas de benefícios, atribuídas aos índices recomendados.

A seguir, são apresentadas algumas métricas sobre custos de processamento, adquiridas do Otimizador do SGBD, bem como, equações para cálculo dos benefícios resultantes dos índices candidatos, desenvolvidas neste trabalho:

(1) Custo de Partida Atual: calculado pelo Otimizador. Faz a estimativa do esforço utilizado antes da varredura de saída iniciar. Esta métrica é gerada antes da criação dos índices hipotéticos e pode ser obtida, também, utilizando o Comando *Explain*, disponível em SGBDs.

(2) Custo Atual: calculado pelo Otimizador, representa o custo total para processamento do comando, baseado no melhor plano de execução sobre a configuração de índices. Esta métrica é gerada antes da criação dos índices hipotéticos e pode ser obtida, também, utilizando o Comando *Explain*.

(3) Custo de Partida Otimizado: calculado pelo Otimizador. Essa métrica faz a estimativa do esforço gasto antes da varredura de saída iniciar, considerando índices hipotéticos e reais na base de dados. Como envolve custo de índices hipotéticos e reais, foi preciso implementá-la neste trabalho.

(4) Custo Otimizado: calculado pelo Otimizador. Essa métrica representa o custo total para processamento do comando, baseado no melhor plano de execução sobre configurações de índices hipotéticos e reais. Como envolve custo de índices hipotéticos e reais, foi preciso implementá-la neste trabalho.

(5) Custo de Criação: representa o custo estimado de criação do índice hipotético, considerando fatores como operações de Entrada e Saída em disco e custos de UCP (Unidade Central de Processamento) [33].

(6) Benefício: essa métrica refere-se a diferença entre o custo do Comando SQL com a configuração de índices reais (Custo Atual) e o seu custo usando a configuração de índices reais e hipotéticos (Custo Otimizado).

(7) Benefício Global: essa métrica refere-se a fração entre o Benefício e o tamanho estimado do índice quando materializado e procura refletir os custos de atualizações que afetam os índices.

Sobre as métricas de custos constantes nos itens listados, vale ressaltar que, como estes

custos são obtidos, sempre para o melhor plano de execução escolhido pelo Otimizador, os valores são sempre positivos ou nulos.

Outro ponto, é que para o cálculo do Benefício Global - BG foi feito um estudo com o objetivo de definir uma métrica, a qual permite ao DBA verificar se um índice hipotético resulta em vantagens ou não para o banco de dados, considerando as atualizações que venham a ocorrer. A equação tem o seguintes elementos:

$$BG = (CA - CO) / T$$

Onde CA corresponde ao Custo Atual e CO corresponde ao Custo Otimizado, descritos nos Itens 2 e 4. T representa o tamanho do índice em *KByte*. Para a obtenção do BG é dividido a diferença do custo atual e custo otimizado pela estimativa do tamanho do índice, que neste trabalho é exatamente igual ao tamanho da tabela. A opção de considerar o tamanho da tabela igual ao índice permite ter uma estimativa conservadora sobre o tamanho real do índice.

É importante ressaltar que para esta métrica, quanto maior o tamanho do índice, pior será o seu benefício global, isto é feito para refletir como as constantes atualizações afetam os índices em ambientes OLTP. Um parâmetro quantitativo para avaliar positivamente o valor de BG é verificar o quanto mais distante ele está do Zero. Nos testes realizados no Capitulo 4, os valores de BG normalmente variaram de 0 a 1.

Para o cálculo do Custo de Criação de um índice, foi utilizada a métrica definida em [33] onde é feito um estudo do código fonte do SGBD PostgreSQL, referente aos cálculos de custos, e definida a seguinte equação:

$$CC = 2P + c R \log R$$

Onde R é o número de tuplas da tabela, P é o número de páginas da tabela e *c* é o coeficiente que relaciona, percentualmente, o custo de uma operação de E/S, com o custo de UCP para processar uma tupla presente na memória. Em nosso estudo de caso, utilizando o SGBD PostgreSQL, cada E/S tem custo estimado igual a 1 e o coeficiente *c* tem valor padrão de 1%.

Supondo um cenário de criação de índice, onde todas as páginas da tabela são lidas, ordenadas e então é criado o índice de forma *bottom-up*, o primeiro termo da fórmula considera o custo de E/S para efetuar leitura de todas as páginas da tabela e escrita de todas as páginas do índice no disco. No segundo termo é feita a estimativa do custo de ordenação de todas as tuplas da tabela em memória.

Vale ressaltar que as estimativas citadas, podem ser exploradas e também adicionadas novas métricas, como por exemplo: cálculo do custo de atualização de índices. Isso exigiria estudos detalhados sobre o modelo de custos dos SGBDs.

3.4 O Algoritmo SIAIO

Para desenvolvimento da Arquitetura para Seleção de Índices Baseada em Custos do Otimizador, foi desenvolvido o algoritmo denominado SIAIO - Seleção de Índices Automática Integrada ao Otimizador, responsável por coordenar todas as ações para recomendação de índices.

A Figura 3.5, mostra todas as etapas gerenciadas e realizadas pelo algoritmo e o seu enquadramento dentro da arquitetura.

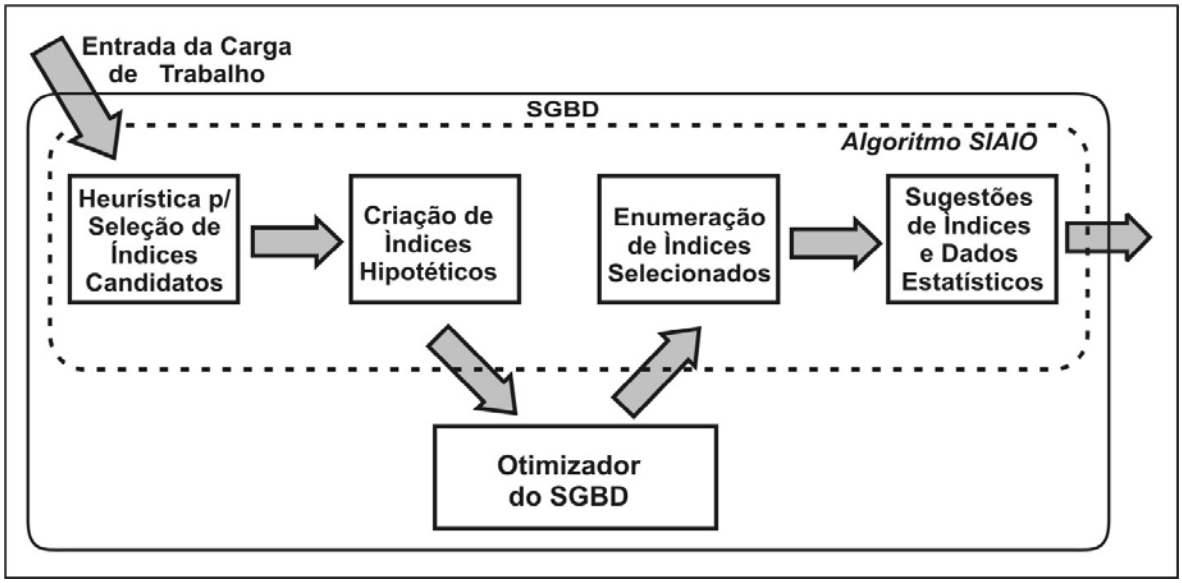

Figura 3.5: Algoritmo SIAIO

Com o algoritmo SIAIO é possível analisar um ou vários Comandos SQL. No entanto, para cada comando é feita uma chamada ao algoritmo, tendo como resultado as sugestões de índices para aquele Comando SQL. Na seqüência é apresentado o fluxo de operações do algoritmo.

1. Inicio: Entrada de uma carga de trabalho W contendo comandos SQL Q_1 ... Q_n, .
2. Para Q_1 Até Q_n Faça
3. A ⟵{Custo total de Q_i sem índice hipotéticos }
4. Verifica em Q_i Se existe cláusula Where. Caso positivo, verifica se existe operadores de agregação, operadores booleanos (AND, OR) e operadores relacionais de intervalos. Então sinaliza as colunas envolvidas.
5. Insere as colunas sinalizadas, na lista de candidatos.
6. Faz a combinação entre os candidatos, afim de encontrar índices de múltiplas colunas, com todas as combinações possíveis e insere estes na lista de candidatos.
7. Verifica em Q_i Se existe cláusula GROUP. Caso positivo, sinaliza as colunas agrupadas.
8. Insere as colunas sinalizadas, na lista de candidatos.
9. Faz a combinação entre as colunas sinalizadas anteriormente, afim de encontrar índices de múltiplas colunas, com todas as combinações possíveis e insere na lista de candidatos.
10. Verifica em Q_i Se existe cláusula ORDER BY. Caso positivo, sinaliza as colunas que serão ordenadas.
11. Insere as colunas sinalizadas, na lista de candidatos.
12. Faz a combinação entre as colunas sinalizadas anteriormente, afim de encontrar índices de múltiplas colunas, com todas as combinações possíveis e insere na lista de candidatos.
13. Para cada elemento da lista de candidatos, implementa um índice hipotético (sem dados).
14. Chamada ao Otimizador, para fazer o planejamento de execução de Q_i, considerando índices reais e hipotéticos.
15. B ⟵{Custo total de Q_i, com índices hipotéticos}
16. Verifica no Plano de Execução, quais índices hipotéticos foram utilizados.
17. Coleta e cálculo de dados estatísticos.
18. Finaliza a análise do comando SQL Q_i . e grava os dados sobre a recomendação dos índices hipotéticos encontrados no plano de execução, em um arquivo tipo texto no disco.
19. Fim Para.
20. Finaliza o algoritmo.

O Algoritmo SIAIO traz como uma das principais contribuições, o desenvolvimento de uma heurística para escolha de índices candidatos, que explora grande parte do espaço de busca de possibilidades existente. A geração de um conjunto de índices relevantes pela Heurística é primordial em uma arquitetura de seleção de índices baseada em custos do Otimizador, porque somente com bons índices candidatos o Otimizador do SGBD conseguirá fazer boas escolhas de índices. Segue no apêndice A a sua implementação na linguagem C.

Não foi desenvolvido neste trabalho, limitação por tempo para percorrer o espaço de busca das possibilidades, recurso que poderia ser necessário para ambientes de grande porte, contendo tabelas no banco de dados, com centenas de colunas. Também não foi implementada a limitação de escolha de índices baseada no espaço em disco disponível. No entanto, a arquitetura desenvolvida permite que esses recursos sejam facilmente adicionados.

CAPÍTULO 4

ESTUDO DE CASO NO POSTGRESQL

Neste capítulo é aplicada a arquitetura para seleção de índices, em um SGBD Relacional. O objetivo é validar a arquitetura proposta no Capítulo 3 e discutir as alterações necessárias ao SGBD para sua implementação. Para completa compreensão do estudo de caso, nas próximas seções são abordados os principais conceitos do SGBD de código aberto PostgreSQL.

Inicialmente na Seção 4.1 é apresentada a arquitetura do SGBD, seu modelo de comunicação, processo de otimização, estimativas de custos e criação de índices. A Seção 4.2 apresenta uma descrição do processo de criação de índices hipotéticos no PostgreSQL. Na Seção 4.3 são apresentadas as alterações efetuadas no código fonte do PostgreSQL, necessárias para viabilização do estudo de caso. Por fim, na Seção 4.4 é apresentada uma visão geral do protótipo desenvolvido, com uma breve execução e avaliação do protótipo.

4.1 Arquitetura do SGBD

Para estudar a seleção de índices em SGBDs relacionais, foi procurado entre os SGBDs disponíveis no mercado aqueles que têm seu código fonte disponível e aberto, possibilitando uma avaliação detalhada dos algoritmos utilizados para sua implementação e também a alteração e inclusão de novos módulos funcionais. Dentre os SGBDs com estas características, os mais utilizados são o PostgreSQL [9] e o MySQL [25]. Para essa implementação foi escolhido o PostgreSQL, principalmente pelas seguintes características: (1) possuir mais recursos que o MySQL e (2) ser direcionado a ambientes OLTP, ao contrário do MySQL, que é direcionado a ambientes WEB.

O PostgreSQL é um sistema gerenciador de banco de dados objeto-relacional, desenvolvido na *University of California at Berkeley*, com versão estável desde 1996. Possui uma comunidade de

desenvolvedores ativa, distribuída mundialmente, encarregada da sua evolução e resolução de problemas encontrados [30].

Desenvolvido na Linguagem de programação C [17], o PostgreSQL é compatível, atualmente, com os principais sistemas operacionais, como: Linux, UNIX e Windows. Possui as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) e vários recursos comuns aos tradicionais SGBDs comerciais, como índices *(index)*, gatilhos *(triggers)*, visões *(views)*, chaves estrangeiras *(foreign keys)* e controle de concorrência. Atualmente o PostgreSQL é conhecido como o SGBD de código aberto mais avançado e com o mais completo conjunto de recursos [9].

4.1.1 Comunicação entre Processos no PostgreSQL

O modelo de comunicação utilizado pelo SGBD PostgreSQL é o cliente-servidor. A arquitetura cliente-servidor do SGBD funciona com um único processo denominado *Postmaster*, localizado no lado servidor, responsável por receber conexões dos clientes. Este processo é responsável pela alocação de memória compartilhada, utilizada por todos os processos e pela comunicação entre os mesmos, no servidor.

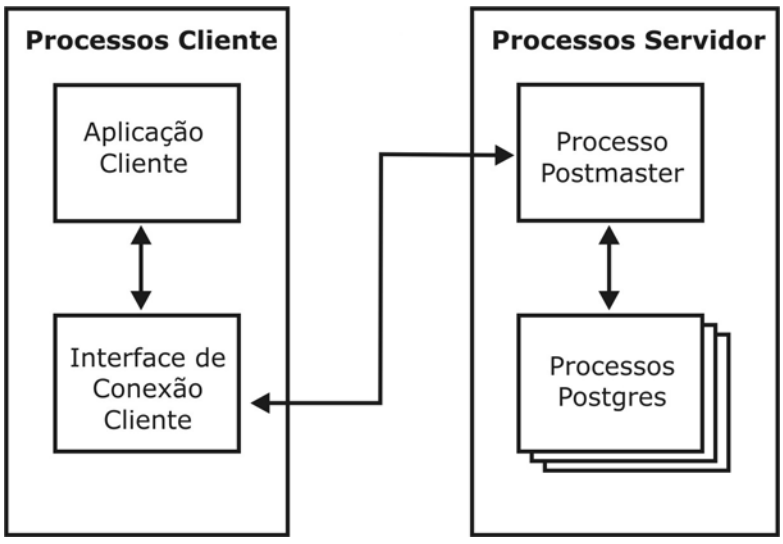

Figura 4.1: Arquitetura interna PostgreSQL

Ao receber conexões clientes, o processo *Postmaster* instancia para cada cliente, um processo denominado *Postgres*. A principal função deste processo é tratar os comandos enviados. Outras atribuições do processo *Postgres* é a manutenção de estatísticas e a gravação física de dados de tabelas e *log*. Para comunicação entre processos *Postgres* são utilizados semáforos e memória compartilhada, garantindo assim, a integridade dos dados mesmo trabalhando, simultaneamente, com comandos enviados por distintos clientes. A Figura 4.1 ilustra o processo de comunicação entre cliente e servidor.

4.1.2 Processamento de Consultas no PostgreSQL

O processamento de consultas no SGBD PostgreSQL, desde o fornecimento do comando SQL por um cliente até o resultado final, é composto pelos passos, constantes na Figura 4.2 [20].

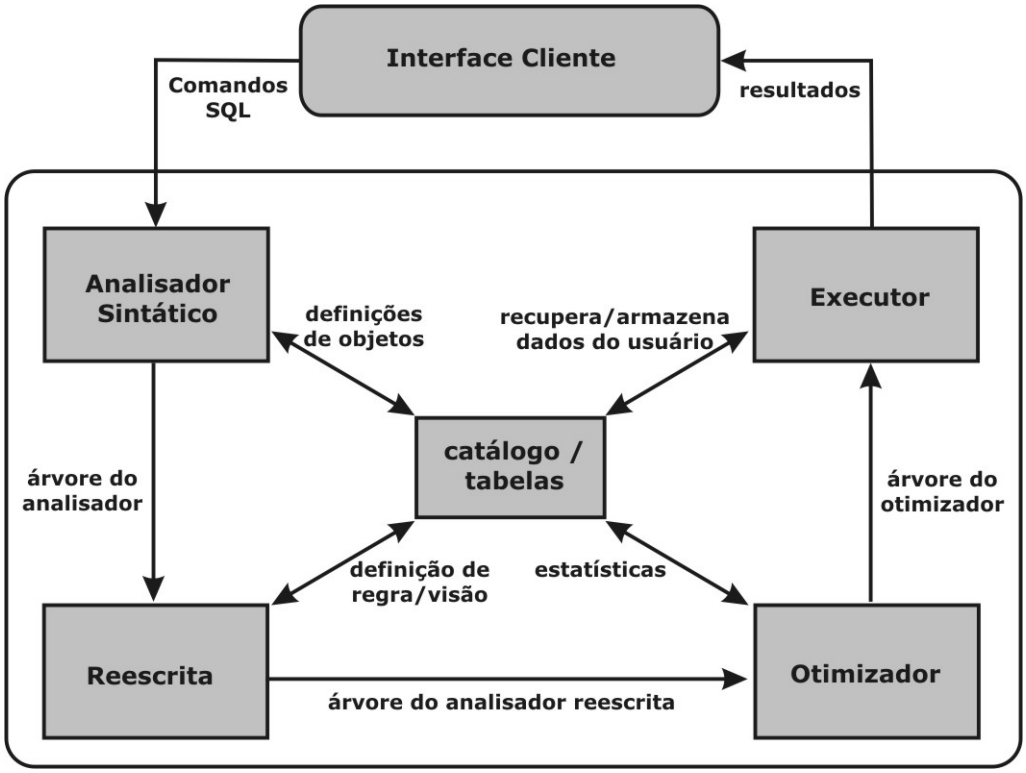

Figura 4.2: Processamento de Consultas no PostgreSQL

Quando um processo *Postgres* recebe um comando SQL de um cliente, o mesmo faz uma chamada ao analisador sintático do SGBD responsável por avaliar a sintaxe do comando, pela geração de uma árvore referente ao comando submetido e pela verificação se o comando submetido é do tipo utilitário ou complexo. Caso o comando seja utilitário, este é executado imediatamente pelo módulo Executor, sem passar por transformações adicionais. Exemplos de comandos utilitários são os da linguagem de definição de dados, como *create table, create index,* etc.

Caso o comando seja do tipo complexo, é necessária a chamada do Módulo de Reescrita, para que seja feita as transformações necessárias antes da execução. Como exemplos de comandos complexos temos *insert, update, delete* e *select*.

O Módulo de Reescrita realiza a análise semântica da árvore gerada, verificando se as tabelas, atributos e funções constantes no comando, realmente existem no banco de dados. Nessa fase são aplicadas as regras de reescrita e restrições de tipos que são armazenados no catálogo do sistema. Como exemplos de reescrita temos a simplificação de funções, transformações de sub-consultas e visões *(views)* em tabelas, entre outras.

Com a árvore semanticamente validada, no Otimizador é gerado todos os caminhos possíveis na árvore, através de uma busca quase exaustiva, para obter as melhores combinações de acesso e operadores físicos, com objetivo de gerar um plano ótimo.

O Otimizador do PostgreSQL utiliza dois algoritmos distintos para geração dos possíveis planos de execução. Para consultas contendo até 11 tabelas é utilizado o algoritmo de programação dinâmica, que utiliza a técnica de buscas exaustivas com um método de corte para determinar a solução ótima. Caso o número de tabelas ultrapasse o limite estabelecido, é utilizado um algoritmo que utiliza uma heurística de busca baseada em algoritmos genéticos.

Com o plano ótimo de execução gerado pelo Otimizador, juntamente com estimativas de custos das operações, é chamado o módulo de Execução do PostgreSQL, que se encarrega de recuperar as tuplas desejadas e as envia ao cliente que solicitou a execução do comando.

4.1.3 Estimativas de Custos

O PostgreSQL, através do comando *Explain*, permite a visualização do plano de execução para cada comando SQL enviado pelos usuários, salvos comandos utilitários (ver seção 4.1.2). Embora os custos fornecidos pelo *Explain* sejam apenas estimativas, já é possível fazer melhorias em comandos e a verificação da utilização de índices por parte do Otimizador do SGBD.

Os dados apresentados pelo comando *Explain* são:

- Custo de partida estimado: estima o esforço gasto antes da varredura de saída iniciar. Como por exemplo, o tempo para fazer a ordenação em um nó.
- Custo total estimado: calcula o custo total gasto, caso todas as linhas forem pesquisadas pelo comando.
- Número de linhas de saída estimado: calcula o número estimado de linhas para o nó envolvido, caso seja executado até o fim.
- Largura média estimada: medida em *bytes*, referente ao tamanho das linhas de saída para o presente nó do plano.

Os custos são medidos em termos de unidades de páginas pesquisadas no disco. O esforço da UCP estimado é convertido em unidades de página de disco utilizando fatores estipulados, altamente arbitrários. Segue um exemplo sobre a Tabela `imóvel` (Figura 4.6).

```
bdimob=# explain select * from imovel order by id_imovel desc;
                              QUERY PLAN
-----------------------------------------------------------------------
 Sort  (cost=1854.39..1879.39 rows=10000 width=167)
     Sort Key: id_imovel
     -> Seq Scan on imovel(cost=0.00..350.00 rows=10000 width=167)
(3 rows)
```

Sobre a leitura dos dados apresentados, é importante ressaltar que o custo, do nível mais alto, inclui todos os custos de seus descendentes. Outro ponto a ser destacado, é que o os custos apresentados, refletem apenas estimativas de execução e não dados reais. No comando *Explain* não é considerado fatores externos ao SGBD, como por exemplo: o tempo para transmissão dos resultados em uma LAN *(Local Area Network)*.

4.1.4 Criação de Índices no PostgreSQL

O PostgreSQL possibilita a implementação de índices utilizando vários tipos de estruturas de dados e organizações definidas pelo próprio usuário. As estruturas de dados permitidas para criação de índices no SGBD são *B-tree*, *Hash* e GIST *(Generalized Search Tree)* [12].

O PostgreSQL utiliza, por padrão, índice em árvore B+, exceto quando não é indicado outra estrutura no comando de definição de índice. Os índices em árvores B+ são os mais indicados na maioria dos casos, principalmente pelo seu desempenho em comandos que utilizam operadores lógicos como: "<", "<=", "=", ">=", ">". No PostgreSQL a definição de índices de múltiplas colunas só é permitida para índices em árvore B+ e GIST.

A estrutura de dados GIST ou Árvore de busca genérica, é uma estrutura de dados e API exclusiva do PostgreSQL, que pode ser utilizada na construção de índices utilizando a maioria das árvores de busca sob vários tipo de dados [12]. GIST pode ser utilizada eficientemente para qualquer tipo de dados que possa ser ordenado em uma hierarquia de conjuntos.

Com GIST é possível criar aplicações com demandas muito específicas, permitindo a

gerência do *layout* das próprias páginas de índices e dos algoritmos de busca e remoção dos índices. Por ser uma estrutura exclusiva do SGBD PostgreSQL e pela necessidade de configurações adicionais para seu funcionamento, optou-se pela não utilização desta estrutura no presente trabalho.

4.2 Índices Hipotéticos no PostgreSQL

Foram implementadas, na presente dissertação, extensões para utilização de configurações de índices hipotéticos no SGBD Objeto-Relacional PostgreSQL [29]. Para isso, foi preciso implementar um mecanismo para armazenamento dos índices hipotéticos no catálogo e estender o Otimizador de consultas, para reconhecer as configurações hipotéticas inseridas. Na Seção 4.3 são apresentadas em detalhes, estas modificações nos componentes do SGBD.

Na implementação do trabalho, a utilização de índices foi restringida a índices completos, organizados em Árvore B+. A escolha por índices completos não tem impacto na implementação, pois índices parciais, somente resultam em vantagens de economia de espaço no armazenamento físico. Já a escolha por índices organizados em Árvore B+, deve-se a sua eficiência e a possibilidade de construção de índices em múltiplas colunas.

4.3 Extensões no Servidor

O PostgreSQL é um SGBD de código aberto, bem organizado e documentado. Todos os códigos relacionados a criação de índices e a máquina de otimização de consultas encontram-se em um único diretório, na estrutura de arquivos que compõem o código fonte, chamado *backend*, assim estruturado:

```
-src
     -backend
              -access
              -bootstrap
              -catalog
              -commands
              -executor
              -lib
              -libpq
              -main
              -nodes
              -optimizer
              -parser
              -po
              -port
              -postmaster
              -regex
              -rewrite
              -storage
              -tcop
              -utils
```

Para implementação da arquitetura foi necessário incluir um módulo para armazenar o Algoritmo SIAIO, dentro do SGBD. Para isso, foi criado um subdiretório denominado "siaio" em src/backend/. É importante ressaltar, que para o funcionamento do algoritmo foram necessárias algumas alterações no SGBD. Embora pequeno o número de alterações, essas são pré-requisitos, pois desta forma, o SGBD irá permitir a definição de estruturas hipotéticas necessárias, bem como permitir que o Otimizador faça estimativas de custos considerando essas estruturas.

Os arquivos do código fonte do PostgreSQL, que precisaram ser modificados foram três. Segue os nomes e localização:

- src/backend/utils/misc/guc.c
- src/backend/optimizer/util/plancat.c
- src/backend/tcop/postgres.c

A primeira alteração foi no arquivo *guc.c*, onde foi adicionado a variável global no SGBD denominada habilitar_siaio, responsável por ativar ou desativar o algoritmo de seleção de índices. Segue na Figura 4.3, o trecho de código incluído.

```
{

     {"habilita_siaio",PGC_USERSET,QUERY_TUNING_OTHER,

      gettext_noop("Habilita algoritmo SIAIO"), NULL
     },
     &habilita_siaio,
     false,
     NULL,
     NULL
}
```

Figura 4.3: Alterações no arquivo *guc.c*

A segunda alteração foi no arquivo *plancat.c*, responsável por rotinas de acesso a informações contidas no catálogo do SGBD, foi inserida uma estrutura de desvio condicional para o provimento de estatísticas para índices hipotéticos. A Figura. 4.4 mostra a inclusão.

```
// a função"verifica_se_ih" determina se o índice é  hipotético ou não, Se indice não é
// hipotético, Então coleta estatistica do índice real, caso o  indice seja hipotético,
// Então recebe dados estatistico da tabela vinculada a ele.
if ( !verifica_se_ih (info->indexoid, &info->pages) )
     info->pages= RelationGetNumberOfBlocks(indexRelation);
else
     info->pages = indexRelation->rd_rel->relpages;
endif
```

Figura 4.4: Alterações no arquivo *plancat.c*

As últimas alterações foram no arquivo *postgres.c*, responsável por chamar o Otimizador do

SGBD. Neste arquivo foram realizadas três alterações (Figura 4.5). A primeira modificação, Figura 4.5 (a), cria uma variável que armazena uma cópia do Comando SQL, para ser utilizada pelo Otimizador, quando o Algoritmo SIAIO estiver em execução. Já a segunda alteração, Figura 4.5 (b), faz à atribuição do Comando SQL, em formato de árvore, para a nova variável criada. Na terceira alteração, Figura 4.5 (c), foi inserida uma estrutura de desvio condicional para possibilitar fazer chamadas ao algoritmo SIAIO durante o processo de otimização.

```
// variável  "queryCopy"  criada para armazenar  uma cópia de comando SQL.
pg_plan_query(Query *querytree, ParamListInfo

boundParams)
{
     Query *queryCopy;
}
```

(a)

```
// Verifica se o algorítmo SIAIO está habilitado
if (pg_siaio)
{
     queryCopy = copyObject( querytree );
}
```

(b)

```
// Verifica se o algorítmo SIAIO está habilitado e faz chamada ao mesmo,
// passando como um dos parâmetros, o comando SQL submetido.
if (pg_siaio)
{
siaio(queryCopy, false, 0, boundParams, plan, false,
      NULL, NULL, NULL);
}
```

(c)

Figura 4.5: Alterações no arquivo *postgres.c*

Além da alteração desses três arquivos, para implementação da arquitetura proposta, foi inserido um arquivo no *backend* do SGBD, denominado *siaio.c*, que contém o algoritmo

propriamente dito, com todas as estruturas de dados e funções necessárias para seleção de índices.

4.4 Visão Geral do Protótipo

Para vislumbrar a utilização da arquitetura para seleção de índices, baseada em custos do Otimizador e realizar a validação do algoritmo implementado, foi criada uma base de dados teste, com algumas tabelas, e a possibilidade de se inserir milhares de registros.

Na Figura 4.6, é apresentado o esquema do banco de dados de uma aplicação do setor imobiliário. Na tabela da esquerda, denominada `Imovel`, são registradas todas as características dos imóveis disponíveis ou não para locação.

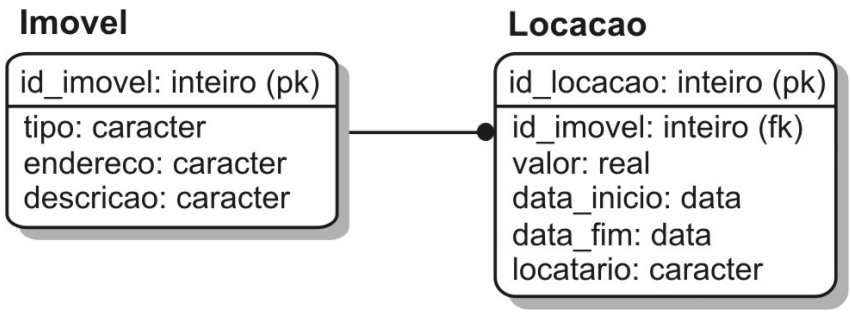

Figura 4.6: Esquema de uma Aplicação Imobiliária

A Tabela `Locacao`, tem o registro de todas as locações realizadas dos imóveis. Essas locações podem estar em andamento ou finalizadas. Para popular as duas tabelas do esquema, foi desenvolvido um programa gerador de dados, utilizando a linguagem de programação C. O programa gerou para a Tabela `Imovel` 10.000 registros. No caso da Tabela `locacao`, foram gerados 200.000 registros. Com objetivo de ter uma base de dados consistente e próxima de um sistema real, foram estabelecidas algumas diretivas, como segue

- As locações se concentraram entre os anos de 1995 a 2008;
- Para cada semana foram geradas aproximadamente 500 locações; e
- Para cada locação foi escolhido aleatoriamente um imóvel.

Para verificar o funcionamento do algoritmo SIAIO, são apresentados alguns exemplos de utilização no SGBD PostgreSQL. Para execução dos comandos, nesta avaliação, é utilizado o utilitário disponibilizado pelo próprio SGBD denominado PSQL. O PSQL é uma ferramenta que permite a inserção de comandos aceitos pelo SGBD e a visualização de seus resultados. Com o PSQL em execução e autenticado no banco de dados da Figura 4.6, o primeiro passo foi habilitar o algoritmo de seleção de índices, através do comando:

```
habilita_siaio = true;
```

Depois de habilitado, o algoritmo SIAIO trabalha em segundo plano. Assim é possível utilizar o SGBD, normalmente, submetendo consultas, fazendo inserções e, atualizações de dados, entre outras funcionalidades. Executando em segundo plano, sem mostrar nenhuma mensagem, o Algoritmo SIAIO é acionado todas as vezes que um comando SQL é submetido ao SGBD. O comando é analisado e, caso haja necessidade, são recomendados índices e gerado dados estatísticos sobre esses. Os resultados gerados pelo algoritmo, ou seja, suas recomendações de índices podem ser vistas no arquivo "sugestoes_indices.txt" localizado no diretório /usr/local/pgsql/pgdata/ do SGBD. Para melhor visualização do trabalho do algoritmo, suponha uma consulta que faz a junção das duas tabelas do esquema e a ordenação dos dados, como segue:

```
bd_imob=# select i.tipo,i.endereco,l.valor,l.locatario
bd_imob=# from imovel i, locacao l
bd_imob=# where valor between 1000 and 3000
bd_imob=# and data_inicio >= '20070101'
bd_imob=# and i.id_imovel = l.id_imovel
bd_imob=# order by l.valor;

<< ENTER >>
```

```
----------------------- RESULTADOS -------------------------
    tipo    |       endereco     |  valor  |    locatario   |
------------+--------------------+---------+----------------
Apartamento | R.Com. Araujo, 105 | 550,00  | Fulano da Silva
Apartamento | R.Mariano Torres,95| 1500,99 | Sicrano da Silva
Residência  | R.XV de Novembro,99| 2000,00 | Beltrano Silva
Residência  | R.Dr.Muricy, 203   | 2550,00 | Fulano M  Silva
Sitio       | Rod. Br 376 km 31  | 2600,00 | Sicrano M  Silva
Residência  | R.Serra Tabatinga,1| 2800,00 | Beltrano M Silva
Apartamento | R.Cons.Laurindo, 63| 3000,00 | Wendel G Pedrozo
...
```

Como visto, o SGBD fez a execução da consulta e retornou os dados. Trabalhando em segundo plano o algoritmo analisou o comando e recomendou índices e estatísticas sobre os mesmos, que podem ser vistas no conteúdo do arquivo "sugestoes_indices.txt" como segue:

```
Comando Submetido:  select i.tipo,i.endereco,l.valor,l.locatario
                    from imovel i, locacao l
                    where valor between 1000 and 3000
                    and data_inicio >= '20070101'
                    and i.id_imovel = l.id_imovel;
                    order by l.valor;
Custo de Partida Atual: 9159.58
Custo Atual: 9205.01
Custo de Partida Otimizado: 6644.89
Custo Otimizado: 6690.31
Tamanho: 6096 kb
Custo de Criação: 10602.05
Benefício: 2514.69
Benefício Global: 0.41
Ganho de desempenho: 29.42%
Coluna(s): valor, data_inicio
Comando SQL:   create index indice_locacao on locacao using
               btree(valor, data_inicio);
```

O conteúdo do arquivo descrito, tem alguns dados e estatísticas (ver Capítulo 3, Seção 3.3) sobre os índices recomendados. Com base nesses dados, um Administrador de Banco de Dados pode verificar o quanto de vantagens um índice recomendado trará para o comando, e para o SGBD como um todo, para só então decidir pela criação ou não do índice.

No exemplo, o Algoritmo SIAIO recomendou a criação de um índice. Para analisar melhor a recomendação, é verificado o plano de execução do Otimizador, elaborado para a consulta. A primeira análise é sem a presença de um índice e após com a presença dele. Para visualizar essas informações, é utilizado o comando *EXPLAIN* do SGBD e também uma representação gráfica gerada pelo software utilitário pgAdmin3. A seguir tem-se o plano de execução do comando SQL sem a presença de um índice.

```
bd_imob=# explain
bd_imob=# select i.tipo,i.endereco,l.valor,l.locatario
bd_imob=# from imovel i, locacao l
bd_imob=# where valor between 1000 and 3000
bd_imob=# and data_inicio >= '20070101'
bd_imob=# and i.id_imovel = l.id_imovel
bd_imob=# order by l.valor;
```

```
                                QUERY PLAN
-----------------------------------------------------------------------
Sort (cost=9159.59..9205.01 rows=18171 width=63)
   Sort Key: l.valor
   -> Hash Join (cost=375.00..7156.55 rows=18171 width=63)
        Hash Cond: (l.id_imovel = i.id_imovel)
      -> Seq Scan on locacao l  (cost=0.00..5037.00 rows=18171 width=8)
           Filter: ((valor >= 1000) AND (valor <= 3000) AND (data_inicio
           >= '2007-01-01 00:00:00-02'))
      -> Hash (cost=350.00..350.00 rows=10000 width=63)
          -> Seq Scan on imovel i(cost=0.00..350.00 rows=10000 width=63)
(8 rows)
```

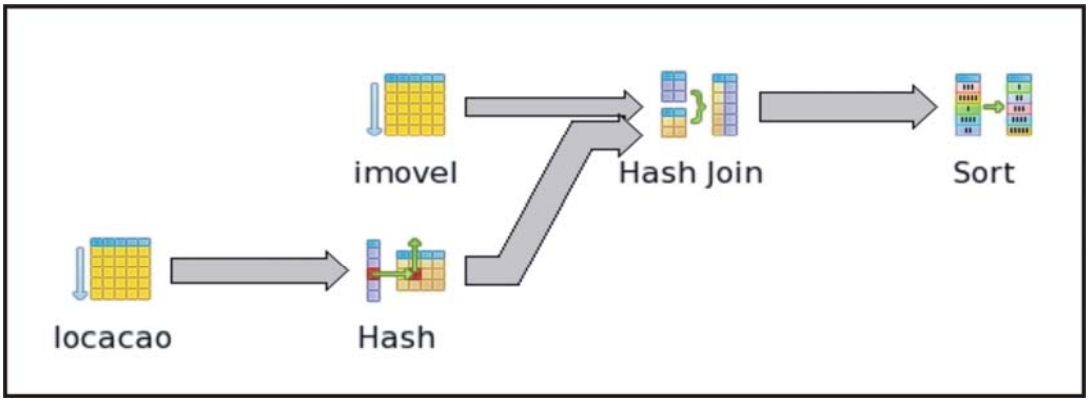

Figura 4.7: Plano para execução de consulta, sem índices

Na análise dos dados gerados, é possível verificar que o SGBD faz uma varredura seqüencial sobre a Tabela `imovel` armazenando os dados em uma tabela *Hash* na memória. Então é realizada uma varredura seqüencial na Tabela `locacao` e aplicado os filtros de `valor` e `data_inicio`. Para cada item de `imovel` é feita uma comparação com as Tabelas *Hash* em memória, contendo dados da Tabela `locacao`. A junção das tabelas é realizada utilizando o Algoritmo *Hash Join* e por fim o resultado é ordenado pelo campo `valor`. O custo estimado final para realizar o processamento da consulta foi de 9159.59. . 9205.01.

Com as informações do plano de execução, sem índices nas tabelas, nesta etapa é materializado o índice sugerido pelo Algoritmo SIAIO, através do Comando SQL encontrado no conteúdo do arquivo de sugestões de índices.

```
bdimobil=# create index indice_locacao on locacao using
          btree (valor,data_inicio);
```

Uma vez materializado o índice real, é verificado o plano de execução que o Otimizador do SGBD escolheria para a consulta, com a presença de um índice real na base de dados.

```
bd_imob=# explain
bd_imob=# select i.tipo,i.endereco,l.valor,l.locatario
bd_imob=# from imovel i, locacao l
bd_imob=# where valor between 1000 and 3000
bd_imob=# and data_inicio >= '20070101'
bd_imob=# and i.id_imovel = l.id_imovel
bd_imob=# order by l.valor;
```

```
                          QUERY PLAN
-----------------------------------------------------------------------
Sort (cost=6453.38..6497.29 rows=17563 width=63)
  Sort Key: l.valor
  -> Hash Join (cost=1502.93..4522.17 rows=17563 width=63)
      Hash Cond: (l.id_imovel = i.id_imovel)
      -> Bitmap Heap Scan on locacao l (cost=1127.93..2972.28
                                        rows=17563 width=8)
            Recheck Cond: ((valor >= 1000) AND (valor <= 3000) AND
                          (data_inicio >= '2007-01-01 00:00:00'))
            -> Bitmap Index Scan on indice_locacao  (cost=0.00..1127.93
               rows=17563 width=0)
                  Index Cond: ((valor >= 1000) AND (valor <= 3000) AND
                               (data_inicio >= '2007-01-01 00:00:00'))
      -> Hash (cost=350.00..350.00 rows=10000 width=63)
          -> Seq Scan on imovel i(cost=0.00..350.00 rows=10000 width=63)
(10 rows)
```

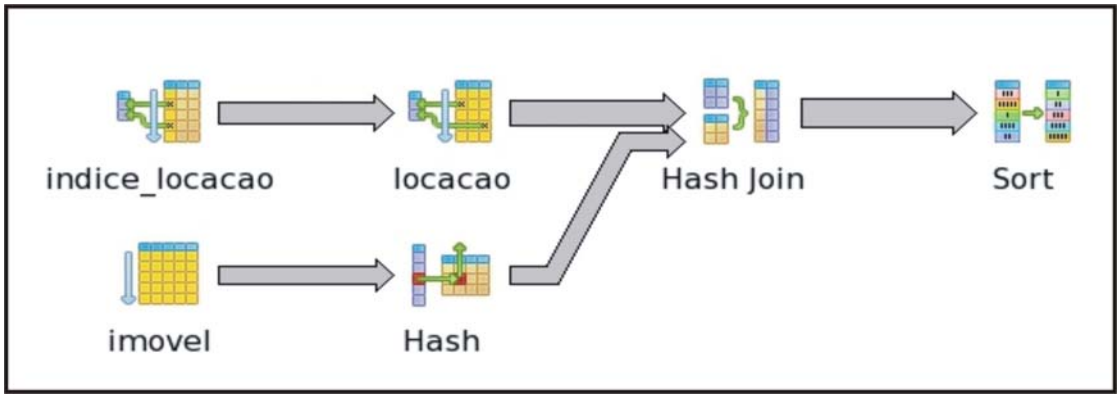

Figura 4.8: Plano para execução de consulta, com índices

Pode-se verificar, que no plano de execução, inicialmente é feita uma varredura seqüencial na Tabela `imovel`, armazenando esta em uma Tabela *Hash* na memória. Com a presença do índice sobre a Tabela `locacao`, o Otimizador faz uma varredura no índice, ao custo de 0.00..1127, já aplicando os filtros de `valor e data_inicio`. Após recuperar os dados da tabela, estes são armazenados em uma Tabela *hash* na memória. Para cada item de `locacao` é feita uma comparação com as Tabelas *Hash*, contendo dados de `imovel`, no intuito de fazer a junção das

tabelas, utilizando o Algoritmo *Hash Join.* Ao final o resultado é ordenado pelo campo `valor`. O custo estimado final para realizar o processamento da consulta foi de 6453.38..6497.29.

A criação do novo índice fez com que o Otimizador, escolhe um plano de execução, diferente do usado no cenário sem índice. O custo de partida foi reduzido em 29.55 % e o custo total foi reduzido em 29.42 % em relação ao cenário sem índice. Portanto, a criação do índice recomendado resultou em maior desempenho ao banco de dados como um todo.

CAPÍTULO 5

AVALIAÇÃO DE DESEMPENHO

Neste capítulo é apresentada a avaliação da arquitetura para seleção de índices, em um ambiente contendo uma base de dados e carga de trabalho, típicos de Ambientes OLTP. O objetivo é verificar a eficácia dos índices recomendados, utilizando a arquitetura para seleção de índices baseada em custos do Otimizador. Para isso, testes foram feitos, em um ambiente que simula condições reais de uso, submetendo ao SGBD cargas de trabalho estáveis, com consultas e alterações realizadas por múltiplos usuários e em pequenas quantidades de tuplas.

Inicialmente, na Seção 5.1 são apresentados os ambientes de software e hardware para execução dos testes. A Seção 5.2 traz a descrição dos testes realizados e detalhes da simulação realizada em ambiente OLTP. Na Seção 5.3 é apresentada a metodologia utilizada nos testes, e a descrição dos cenários de testes empregados. A Seção 5.4 apresenta os resultados obtidos em cada cenário e o resultado geral do Benchmark. É realizada, na Seção 5.5, uma análise comparativa dos índices, gerados pelo Algoritmo SIAIO com os índices do *Toolkit* DBT-2. Por fim, na Seção 5.6, é apresentada a comparação com trabalhos correlatos.

5.1 Ambiente Experimental

Os experimentos foram realizados em um computador com processador Intel Pentium IV, a 3 GHz, 1 GB de Memória SDRAM e Disco Rígido Serial ATA de 80 GB. O sistema operacional instalado é o Linux Slackware Versão 12.0, *Kernel* 2.6.21.5, com Ambiente Gráfico K Desktop *Environment*. Para execução dos testes, o ambiente foi automatizado através do uso de *Shell Scripts* para o Ambiente *Bash*.

O SGBD utilizado foi o PostgreSQL Versão 8.2.0 *Stable*. Para visualização e execução de comandos SQL em modo gráfico, foi utilizado o Software Utilitário para PostgreSQL pgAdmin3 [28].

5.2 Visão Geral do Benchmark

Para simulação de um Ambiente OLTP, foi utilizado o *Toolkit* DBT-2 *(Database Test 2)*, provido pela Organização OSDL *(Open Source Development Labs)* [26]. Este *Toolkit* disponibiliza *scripts* que permitem a criação automática de um Ambiente OLTP.

A Organização OSDL mantém um conjunto de testes de desempenho para avaliar o comportamento de soluções de software livre, em especial o Sistema Operacional Linux. Esses testes são inspirados nos *Benchmarks* do TPC *(Transaction Processing Performance Council)* [38].

Embora os *benchmarks* providos pela Organização OSDL utilizem os mesmos esquemas, transações e forma de geração de dados dos *Benchmarks* do TPC, os resultados obtidos não podem ser comparados com resultados oficiais publicados pelo TPC. Resultados oficiais publicados pelo TPC aderem a um conjunto de práticas de auditoria que requerem, entre outras informações, a publicação da configuração e preços de todos os produtos utilizados para obtenção dos resultados.

A origem da base de dados e transações constantes no *Toolkit* DBT-2, utilizado neste trabalho, é o *benchmark* TPC-C [38]. O ambiente simulado pelo *benchmark* consiste em um sistema de processamento de transações de uma empresa atacadista que produz, vende e distribui seus produtos. Essa companhia encontra-se geograficamente distribuída, de maneira que suas vendas estão espalhadas por um número de armazéns e distritos. O tamanho da organização é definido pelo número de armazéns que ela possui, conforme ela cresce mais armazéns e distritos precisam ser criados. Cada armazém mantém estoque para todos os cem mil produtos vendidos pela companhia e cada um dos seus distritos atende a três mil clientes. A Figura 5.1 ilustra a hierarquia entre

armazéns, distritos e clientes, segundo [38].

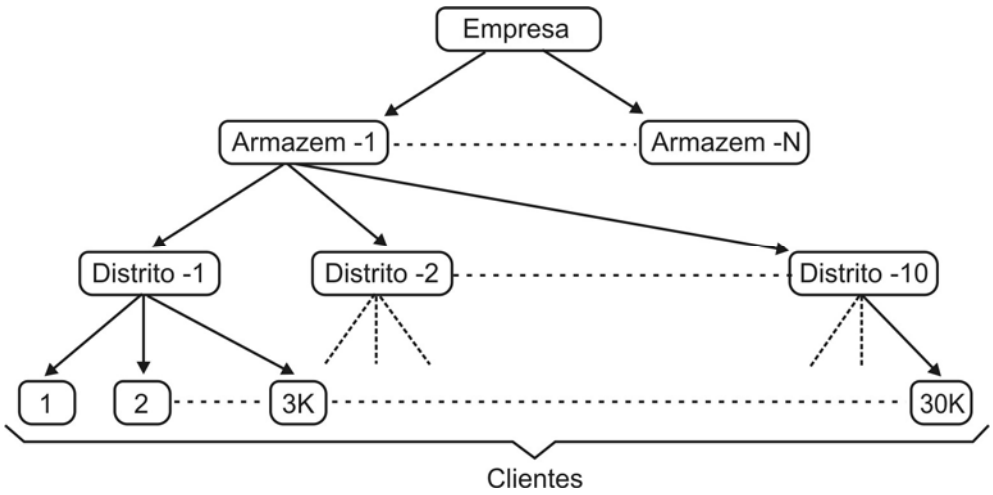

Figura 5.1: Relação entre armazéns, distritos e clientes

O ambiente do *benchmark* simula uma carga de trabalho, composta por um conjunto de cinco transações de leitura e escrita, como segue:

- Novo Pedido: consiste no registro de uma operação de compra de um cliente. É uma transação executada com uma freqüência alta, que impõe uma carga média ao sistema por meio de operações de leitura e de escrita. Trata-se da principal transação do *benchmark* e tem como resultado o número de transações executadas por minuto.
- Pagamento: consiste no registro do pagamento de uma compra. É uma transação executada com alta freqüência, mas de peso leve e composta por operações de leitura e escrita.
- Entrega: transação que registra a entrega de um pedido. Esta transação faz o processamento em *batch* de dez novos pedidos. É uma transação com baixa freqüência, que impõe uma carga moderada ao sistema por meio de operações de leitura e escrita.
- Situação do Pedido: transação que consulta a situação da última compra de um cliente. É uma transação executada com baixa freqüência, que impõe uma carga moderada ao sistema, através de operações somente de leitura.
- Nível de Estoque: transação responsável por determinar dentre os últimos itens vendidos, quais estão com o estoque abaixo do limite. É executada com baixa freqüência, porém impõe uma carga pesada ao sistema através de operações somente de leitura.

Como abordado, cada transação possui sua freqüência de execução. Na Tabela 5.1, são

mostradas as freqüências, representando qual é a parcela esperada para cada transação no número total de transações executadas pelo *benchmark*.

Tabela 5.1: Freqüência de transações

Nome da Transação	Freqüência de Execução
Novo Pedido	45 %
Pagamento	43 %
Entrega	4 %
Nível de Estoque	4 %
Situação do Pedido	4 %

Durante a execução do benchmark, é simulado clientes fazendo pedidos dos itens mantidos em estoque. É mantido também informações de histórico de compras dos clientes.

5.2.1 Esquema do Banco de Dados

O banco de dados especificado pelo *benchmark* é composto por nove tabelas. A Figura 5.2 mostra essas tabelas e os seus respectivos relacionamentos, segundo [22].

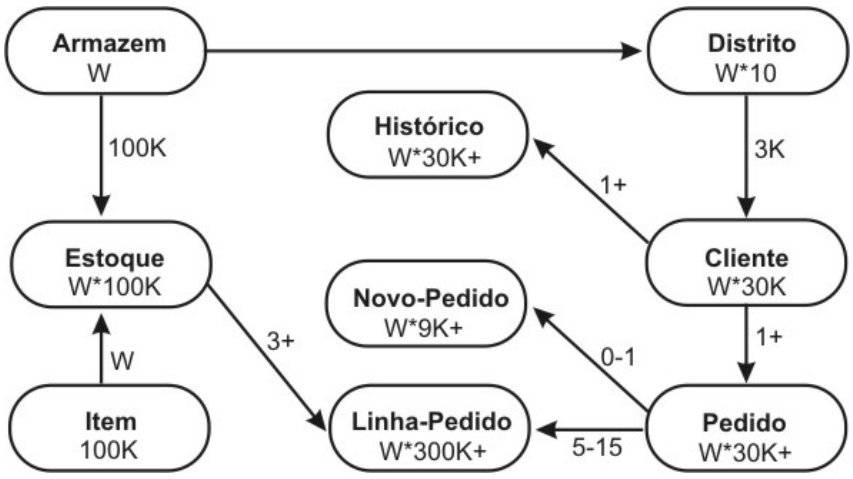

Figura 5.2: Esquema lógico do *benchmark*

Os números próximos às linhas de relacionamento representam a cardinalidade dos relacionamentos. Já os números dentro das entidades indicam a cardinalidade das tabelas (número

de linhas). Por exemplo, W indica que a Tabela `Armazem` é populada com W itens de dados. A Tabela `Distrito`, por sua vez, tem 10*W linhas. Interessante notar que, com exceção da Tabela `Item`, cuja cardinalidade é fixada em 100 mil linhas, todas as outras tabelas têm suas cardinalidades proporcionais a Tabela `Armazem`.

5.2.2 Arquitetura OLTP

Para simular um Ambiente OLTP, o *Benchmark* DBT-2 foi projetado utilizando três grandes componentes (Figura 5.3). Esses componentes são identificados como Terminal Simulador de Usuário (TSU), Cliente WEB e Servidor de Banco de Dados. Esses se comunicam através de uma LAN *(Local Area Network)*, utilizando Protocolo de Rede TCP/IP.

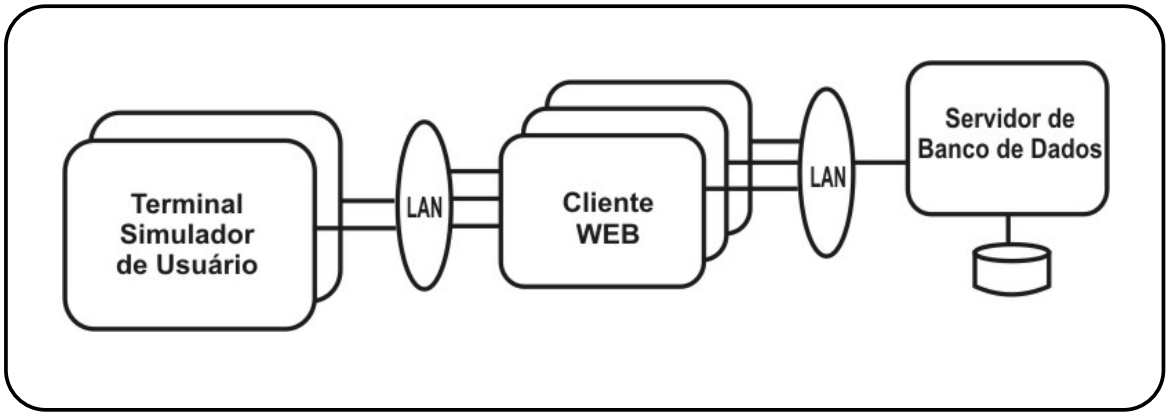

Figura 5.3: Ambiente de Teste DBT-2

Neste ambiente, os TSUs representando usuários que geram carga de trabalho através das chamadas às cinco transações disponíveis. Os Clientes WEB, responsáveis por receber as solicitações, mapeiam diversos clientes a um único Servidor de Banco de dados, que faz o processamento da carga de trabalho.

5.2.3 Métricas de Desempenho e Escala

O *benchmark* utilizado neste trabalho teve como meta verificar a vazão do sistema em termos das transações que executa por unidade de tempo, ou seja, o seu *throughput*. O *throughput* reportado ao

final da execução do *Benchmark* DBT-2 está diretamente relacionado ao número de usuários simulados, que submetem transações ao SGBD e a carga é proporcional ao número de usuários. Por sua vez, segundo as regras do *benchmark*, o número de usuários deve corresponder a dez vezes o número de armazéns do banco de dados. Desse modo, a cardinalidade da Tabela `armazem` é o fator de escala do *Benchmark* DBT-2, determinando a carga submetida ao sistema. Nos testes realizados neste trabalho, a maior escala foi de 3 (três) armazéns, ou seja, a Tabela `armazem` teve no máximo 3 (três) tuplas gravadas.

A métrica de desempenho utilizada pelo *Benchmark* DBT-2 é o tpmC, que mede o número de transações do tipo Novo Pedido completadas por minuto. Apesar da métrica refletir o *throughput* de apenas um único tipo de transação, o processamento das outras transações influência no tempo de resposta da transação Novo Pedido, o que garante que o desempenho do sistema seja testado por meio dessa transação.

5.3 Metodologia e Cenários de Testes

Nesta seção, são apresentados os cenários utilizados nos testes, bem como a metodologia empregada.

Para avaliação da arquitetura para seleção de índices, baseada em custos do Otimizador e comparação dos resultados, foram estabelecidos três cenários de teste:

1. Execução do *Benchmark* DBT-2, sem índices na base de dados. Este é o cenário com menor *throughput* observado, uma vez que todas as consultas são processadas através de varreduras seqüenciais nas tabelas do Banco de dados.
2. Execução do *Benchmark* DBT-2 com os índices sugeridos pelo toolkit. Este cenário permite que o sistema tenha alto *throughput*, pois utiliza os índices recomendados pelos desenvolvedores do *Toolkit* DBT-2.

3. Execução do *Benchmark* DBT-2 com os índices recomendados pelo Algoritmo SIAIO. Neste cenário espera-se que o sistema apresente uma vazão, no mínimo, próxima ao segundo cenário.

O maior tempo de execução do *benchmark* foram três horas, tempo que acredita-se ser suficiente para verificar o comportamento do sistema. As escalas utilizadas no *benchmark* variaram de um a três armazéns. É importante ressaltar, que um *benchmark* executado nessa escala e tempo, oferece uma carga de trabalho similar a de ambientes OLTP.

A metodologia utilizada para os testes apresenta como principal característica, a execução de cada experimento, 10 vezes. A execução do experimento por várias vezes, tem por objetivo obter resultados mais confiáveis, além de evitar discrepâncias entre os mesmos. Para calcular o resultado final entre os 10 experimentos, é aplicada a média aritmética, excluindo o melhor e pior resultado.

Para realização dos testes, utilizando a metodologia descrita anteriormente, foram necessários a execução de 90 experimentos, ou seja, 30 experimentos para cada cenário/escala, com duração de 3 (três) horas cada, totalizando 2.700 horas.

5.4 Resultados Experimentais

Nesta seção, são apresentados alguns resultados obtidos, com a aplicação do *Benchmark* no SGBD PostgreSQL. Os resultados obtidos foram através do *Toolkit* DBT-2. Ao instalar o *toolkit*, esse cria automaticamente alguns índices, para melhor processamento da carga de trabalho e também como conseqüência da criação de chaves primárias.

O gráfico mostrado na Figura 5.4, apresenta a execução do *Benchmark* nos três cenários: Sem Índices, Índices DBT-2 e Índices SIAIO. O *throughput* do sistema foi testado durante três horas, utilizando como fator de escala a variação do número de armazéns. A seguir são discutidos os resultados de cada cenário individualmente.

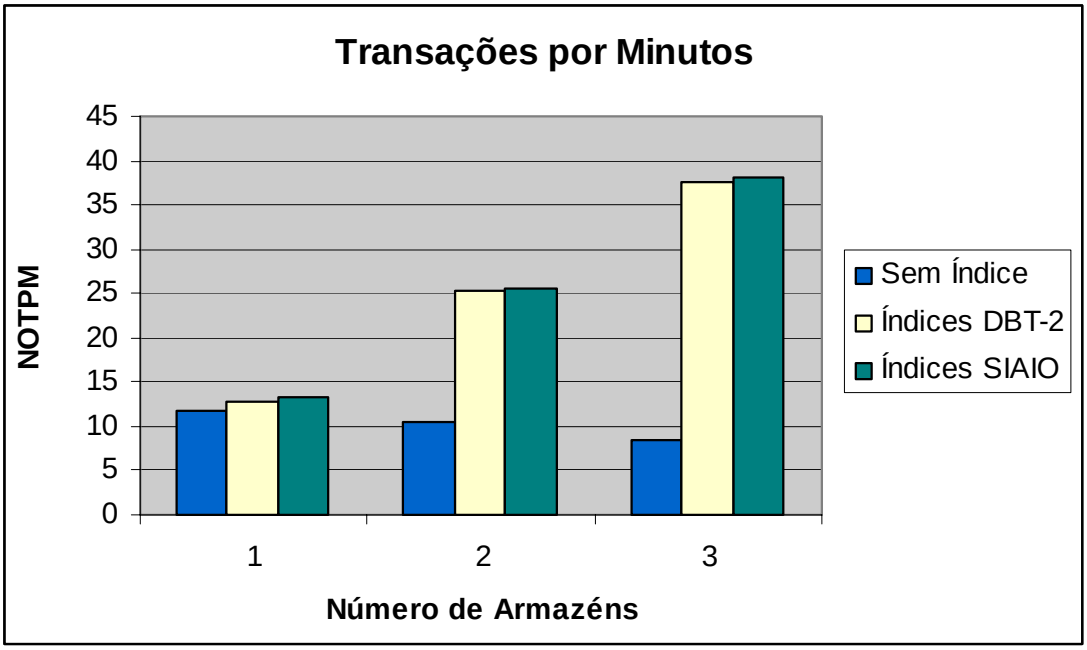

Figura 5.4: Número de Transações por Minuto (NOTPM)

5.4.1 Primeiro Cenário

Neste é verificado o menor *throughput* do *benchmark.* Por não ter nenhum índice, conforme a base de dados aumenta, com o incremento do número de armazéns, ocorre uma diminuição constante no *throughput* apresentado. Isto ocorre, pois em uma base de dados sem índices, as consultas são processadas com varredura seqüencial, o que implica em mais tempo gasto para processar cada comando, à medida que a base cresce.

A ausência de índices na base de dados, além de degradar o desempenho do SGBD, também causa sobrecarga em outras partes do servidor hospedeiro, como: UCP (Unidades Central de Processamento), Memória primária e secundária (discos magnéticos).

Outra desvantagem ocasionada pela ausência de índices é nas operações de entrada e saída. Na Figura 5.5 é possível verificar a taxa de Entrada e Saída em um disco magnético durante o *benchmark*.

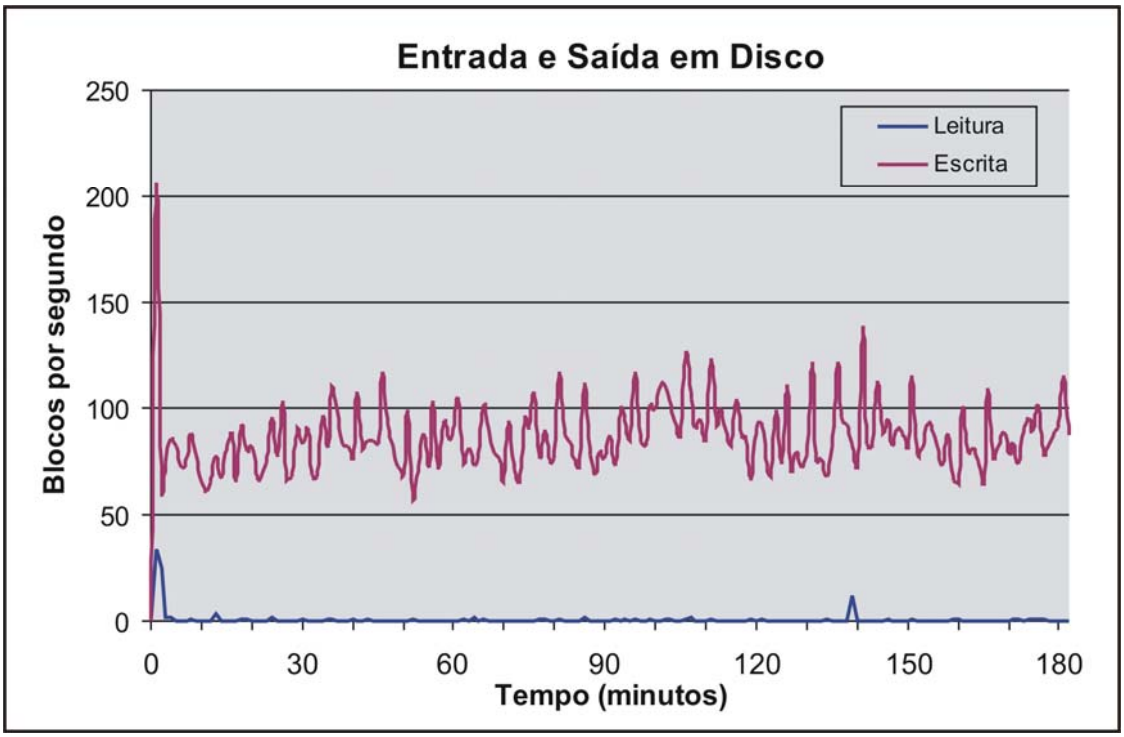

Figura 5.5: Entrada e Saída em Disco para cenário 1

Na Figura, podem-se observar as baixas taxas de leituras, e médias taxas de escrita em disco, para o primeiro cenário. Em uma base de dados sem índices, os discos magnéticos ficam ociosos em determinados períodos, esperando a UCP terminar o processamento. Analisando as estatísticas geradas pelo *Toolkit* DBT-2, observou-se que as baixas taxas de leitura foram causadas pela estratégia utilizada pelo Otimizador do SGBD, no processamento dos comandos. Como não existe índices, a tabela inteira foi carregada para a memória RAM, a qual tem um custo de acesso mais baixo, que o acesso a disco, ocasionando baixas taxas de leitura em disco.

5.4.2 Segundo Cenário

Conforme está na Figura 5.4, neste cenário são utilizados os índices definidos pelo *Benchmark* DBT-2. Em relação ao cenário anterior, ocorre uma melhora significativa na vazão do sistema.. Com a presença de índices na base de dados, a medida que a base de dados aumenta, tem-se mais terminais acessando o SGBD, resultando em mais transações executadas por minuto. O uso de

índices permite que o tempo de resposta permaneça praticamente constante, mesmo quando ocorre crescimento da base.

Outra vantagem é nas operações de entrada e saída em disco. Na Figura 5.6 pode-se verificar que ambos os parâmetros tiveram significativos aumentos. Com a utilização de índices, o acesso a disco é mais freqüente, pois não há sobrecarga da UCP e memória primária, ocasionando um número maior de transações sendo executadas por minuto.

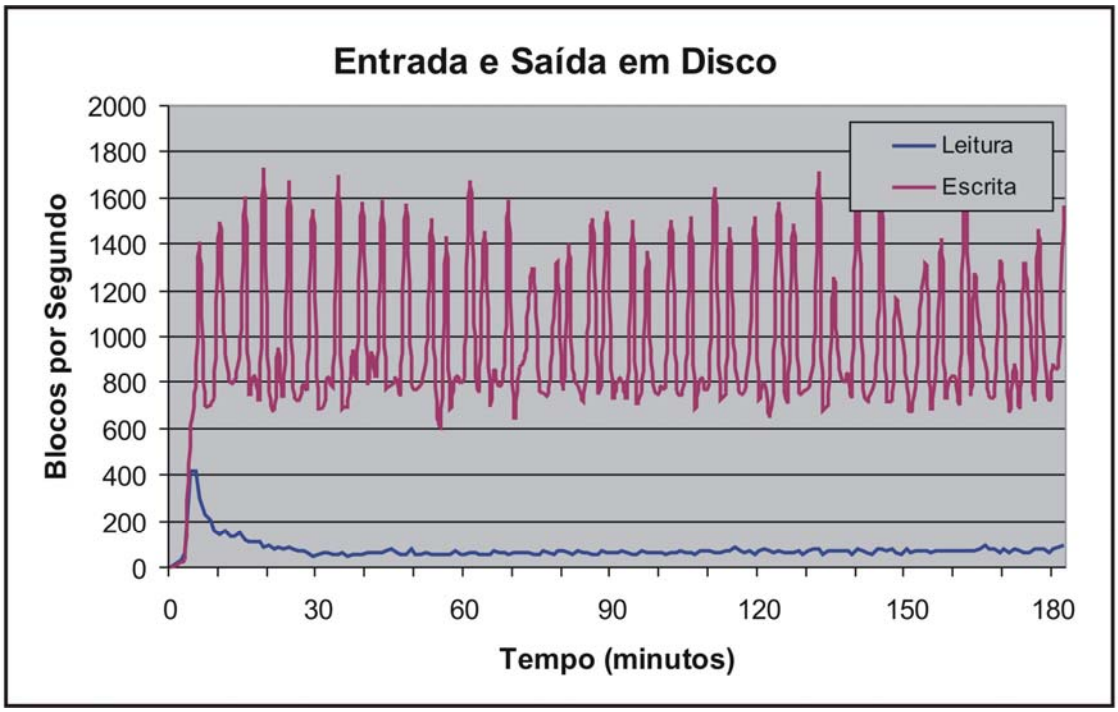

Figura 5.6: Entrada e Saída em Disco para cenário 2

5.4.3. Terceiro Cenário

Para realizar os testes no terceiro cenário, foi necessário gerar os índices utilizando a arquitetura para seleção de índices implementada neste trabalho. Para isso, primeiramente foram retirados todos os índices presentes no banco de dados. Após, o Algoritmo SIAIO foi colocado em execução e então iniciado o *benchmark*. Finalizado o *benchmark*, os índices sugeridos pelo Algoritmo SIAIO foram todos aplicados no SGBD. Com a materialização física dos índices sugeridos pelo Algoritmo

SIAIO, o *benchmark* foi executado novamente.

Na Figura 5.7, é mostrado um gráfico sobre entrada e saída em disco para este cenário. É possível verificar que a taxa de blocos solicitados é muito maior que no primeiro cenário, devido à utilização de índices.

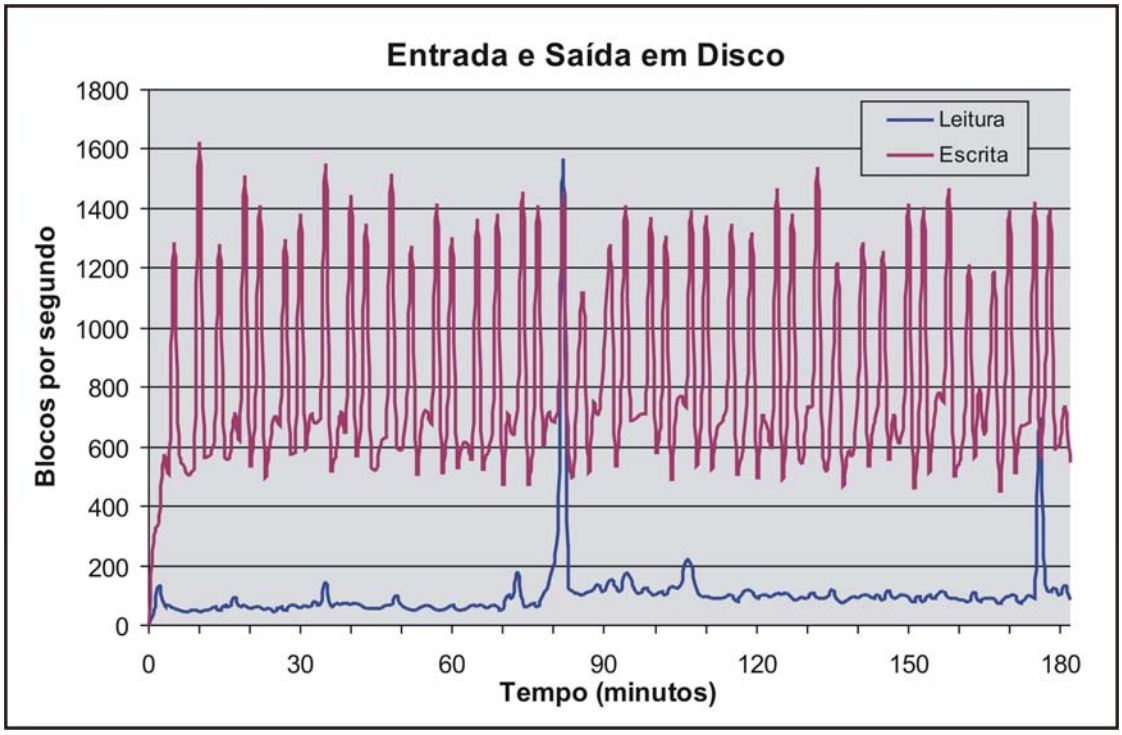

Figura 5.7: Entrada e Saída em Disco para cenário 3

Como esperado, com a presença de índices, o SGBD teve um *throughput* muito superior ao primeiro cenário e um leve ganho em relação ao segundo cenário. Outro ponto a ser comparado é o *throughput* do segundo cenário com o terceiro, figura 5.4. Quanto mais *throughput* tiver o terceiro cenário em relação ao segundo, melhor terá sido o trabalho executado pelo Algoritmo SIAIO. Segue nas tabelas a seguir os resultados obtidos em todos os experimentos. É possível constatar que em todos as escalas, o terceiro cenário apresentou melhor desempenho que o segundo e o primeiro cenários.

Tabela 5.2: Comparativos entre cenários de teste com 1 armazém

Escala - 1 Armazéns			
	Cenário 1	**Cenário 2**	**Cenário 3**
Melhor resultado	11,51	12,87	12,97
Pior Resultado	10,65	12,42	12,71
Experimento 1	11,36	12,77	12,88
Experimento 2	11,32	12,79	12,89
Experimento 3	11,44	12,67	12,91
Experimento 4	11,24	12,85	12,73
Experimento 5	11,22	12,9	12,72
Experimento 6	11,46	12,65	12,81
Experimento 7	11,11	12,5	12,72
Experimento 8	11,57	12,59	12,75
Media	**11,34**	**12,72**	**12,80**

Tabela 5.3: Comparativos entre cenários de teste com 2 armazéns

Escala - 2 Armazéns			
	Cenário 1	**Cenário 2**	**Cenário 3**
Melhor resultado	10,65	25,45	25,73
Pior Resultado	9,51	24,82	25,06
Experimento 1	10,57	25,15	25,35
Experimento 2	10,47	25,19	25,32
Experimento 3	10,69	25,24	25,22
Experimento 4	10,38	25,26	25,2
Experimento 5	10,78	25,29	25,17
Experimento 6	10,55	25,3	25,45
Experimento 7	10,59	25,32	25,45
Experimento 8	10,61	25,33	25,51
Media	**10,58**	**25,26**	**25,33**

Tabela 5.4: Comparativos entre cenários de teste com 3 armazéns

Escala - 3 Armazéns			
	Cenário 1	**Cenário 2**	**Cenário 3**
Melhor resultado	8,98	38,01	38,16
Pior Resultado	8,05	37,46	37,64
Experimento 1	8,31	37,83	37,86
Experimento 2	8,43	37,48	38,01
Experimento 3	8,27	37,98	38,03
Experimento 4	8,47	37,76	37,82
Experimento 5	8,16	37,91	37,83
Experimento 6	8,58	37,93	37,72
Experimento 7	8,35	37,79	37,67
Experimento 8	8,39	37,65	37,88
Media	**8,37**	**37,79**	**37,85**

No apêndice B, são apresentados alguns experimentos constando a média de todas as transações presentes no *Benchmark*, executadas em 3 horas, nos três cenários abordados.

5.5. Análise Comparativa dos Índices

Nesta seção é realizada uma análise comparativa sobre a seleção de índices realizada pelo *Toolkit* DBT-2 e pelo Algoritmo SIAIO. O objetivo é comparar qualitativamente os índices sugeridos pelos desenvolvedores do *benchmark* e os índices conseguidos com a execução do Algoritmo SIAIO.

Há um conjunto de treze índices que foram criados pelo Algoritmo SIAIO, contra dez do *toolkit DBT-2*, conforme pode ser observado na Tabela 5.3. Destaca-se que alguns dos índices são muito semelhantes, apresentando como diferenças, por exemplo, a ordem empregada para as colunas.

Tabela 5.5: Comparativos entre índices

Tabela	Coluna(s) utilizadas - Toolkit DBT-2	Coluna(s) utilizadas - Algorítmo SIAIO
Cliente	c_w_id, c_d_id, c_id)	(c_d_id,c_id)
Cliente	(c_w_id, c_d_id, c_last, c_first, c_id)	(c_w_id,c_d_id, c_last)
Cliente	Nenhuma	(c_d_id)
Distrito	(d_w_id,bd_id)	(d_id)
Item	(i_id)	(i_id)
Novo_pedido	(no_w_id, no_d_id, no_o_id)	(no_d_id)
Novo_pedido	Nenhuma	(no_d_id, no_o_id)
Linha_Pedido	(ol_w_id, ol_d_id, ol_o_id, ol_number)	(ol_d_id,ol_o_id)
Linha_Pedido	Nenhuma	(ol_o_id)
Linha_Pedido	Nenhuma	(ol_d_id)
Pedido	(o_w_id, o_d_id, o_id)	(o_d_id,o_id)
Pedido	(o_w_id, o_d_id, o_c_id)	(o_c_id,o_d_id)
Estoque	(s_w_id, s_i_id, s_quantity)	(s_i_id)
Armazem	(w_id)	Nenhuma

Outra diferença, é o índice indicado pelo *toolkit*, para a Tabela `Armazem`, que não é indicado pelo Algoritmo SIAIO. Como nos experimentos, esta tabela apresenta apenas 3 (três)

registros no máximo, o Algoritmo SIAIO decide não criar índice, o que reflete a premissa da arquitetura baseada em custos do Otimizador, que procura somente selecionar índices que tragam vantagens de desempenho para o SGBD.

Para análise do comportamento dos índices, tem-se na Tabela 5.4 um comparativo dos custos do Otimizador, entre os Cenários 2 e 3. Foram comparados dez comandos SQL da transação Novo Pedido, que é a principal transação do *benchmark.* Para obtenção dos custos, foi utilizado o comando *Explain* do SGBD.

Tabela 5.6: Custos - Transação Novo Pedido

Comandos SQL	**Custos Cenário 2**	**Custos Cenário 3**
Q1	0.00..8.02	0.00..315.56
Q2	4.01..332.35 0.00..4.01	28.15..94.44 0.00..28.15
Q3	4.01..333.14 0.00..4.01	28.15..94.45 0.00..28.15
Q4	0.00..8.04	0.00..23.52
Q5	0.00..0.01	0.00..0.01
Q6	0.00..0.02	0.00..0.02
Q7	0.00..8.03	0.00..8.03
Q8	0.00..8.11	0.00..23.63
Q9	0.00..8.11	0.00..23.64
Q10	0.00..0.02	0.00..0.02
Transações por minuto	62,70	63,71

Os custos gerados pelo comando *Explain*, são uma estimativa separada em dois valores, entre dois pontos consecutivos: o custo de partida estimado e o custo total estimado. Os custos são medidos em termos de unidades de página pesquisadas no disco.

Na Tabela 5.4, o segundo cenário, com índices sugeridos pelo *Toolkit* DBT-2, observa-se que os custos são iguais em 4 (quatro) comandos, melhores em outros 4 (quatro) e piores em 2 (dois) comandos. Outro ponto importante é o número de transações obtido ao final da bateria de

teste, que foi favorável ao Cenário 3, com índices sugeridos pelo Algoritmo SIAIO. O maior número de transações por minuto para o terceiro cenário mostra que, embora na maioria dos comandos os custos tenham sido maiores para este cenário, nos dois comandos mais importantes, Q2 e Q3, os custos do terceiro cenário foram menores. Isto mostra a eficiência da recomendação de índices utilizando a arquitetura baseada em custos do Otimizador implementada nesta dissertação.

5.6 Comparação com Trabalhos Correlatos

Os primeiros projetos de recomendação de índices foram iniciados em meados dos anos 80 [2,3,6, 11,13,24]. Esses são restritos a determinados SGBDs e não utilizam o Otimizador para estimar os seus custos. Um exemplo disso pode ser visto em [6], onde são desenvolvidos modelos de custos detalhados para a ferramenta de seleção de índices. Esses modelos não estão em sincronia com o modelo utilizado pelo Otimizador de consultas do sistema. Isto não é desejável, uma vez que o Otimizador não pode realizar as mesmas suposições sobre custos que a ferramenta e pode causar, até mesmo, a escolha de métodos de acesso distintos dos previstos pela ferramenta, para o processamento das consultas.

O trabalho de Filkestein et. al [10], propõe que ao realizar a seleção de índices para um determinado banco de dados, sejam criadas réplicas, no catálogo do SGBD, das tabelas envolvidas. Essas réplicas não possuem extensão física, mas recebem as informações estatísticas das tabelas originais. A ferramenta, então, cria índices sobre as réplicas e faz estimativas de custos. É proposto, ainda, que o SGBD possua um comando para obtenção do custo e do plano de execução que o Otimizador gera para uma determinada consulta. Este comando, chamado *Explain*, é um recurso atualmente disponível em muitos SGBDs comerciais.

Para obter uma estimativa de custos de execução de uma consulta sob uma configuração hipotética, a ferramenta de seleção de índices troca as referências às tabelas na consulta pelas suas

respectivas réplicas e utiliza o comando *Explain* para estimar métricas de desempenho sobre os índices recomendados.

Uma desvantagem desta abordagem é o fato de que a ferramenta de seleção de índices precisa ter privilégios de acesso suficientes às informações críticas presentes no catálogo sobre estatísticas de otimização e as altere diretamente. Nesta dissertação é adicionado o conceito de índices hipotéticos ao sistema, que permite que uma ferramenta de seleção de índices faça simulações de configurações no SGBD de forma mais organizada do que através da criação de réplicas de tabelas no catálogo.

Em [33] é proposto uma ferramenta que automatiza completamente a seleção de índices em SGBDs relacionais. No trabalho são utilizados agentes de softwares. integrados ao SGBD, encarregados da obtenção da carga de trabalho e da criação automática dos índices. Porém os agentes de software, nesta implementação, devem estar embutidos junto ao código do SGBD, o que torna a ferramenta fortemente dependente de tecnologia, devido a cada SGBD ter suas características, métodos de coleta de estatísticas, projeto físico e outras características.

O trabalho [11] propõe que o Otimizador de consultas do SGBD seja estendido para permitir a obtenção de um plano de execução, sob a suposição de que um dado conjunto de índices hipotéticos exista na base. Porém, não explora quais alterações seriam necessárias no SGBD para dar suporte à noção de índices hipotéticos.

Outras soluções [4,37], utilizam o Otimizador do SGBD para avaliar os custos dos índices recomendados, mas não para recomendar índices. O processo de recomendação é realizado em um módulo externo ao SGBD.

O trabalho [5], implementado no SGBD Microsoft SQL Server, oferece uma importante contribuição. Foram combinadas as vantagens dos algoritmos de recomendação de índices de uma coluna com as dos algoritmos de otimização de múltiplas-colunas. Considerando índices candidatos

com um pequeno número de colunas, há mais possibilidade para otimizar várias consultas usando os mesmos índices candidatos e ainda compactá-los dentro de pequenas restrições de disco. Levando isso em conta, a ferramenta inicia considerando índices de pequeno número de colunas, e trabalha sobre os índices de múltiplas colunas se o tempo permitir. Essa estratégia procura diminuir o número de chamadas ao Otimizador. Entretanto, as mesmas vantagens de reduzir o número de chamadas ao Otimizador podem ser obtidas se for inserido o algoritmo de seleção de índices sobre o Otimizador, como é proposto no presente trabalho, o qual acredita-se ser a melhor técnica para solução do problema de seleção de índices.

Outras vantagens empregada no presente trabalho comparadas ao trabalho de [5] é a recomendação em larga escala de índices, intrínseco na heurística de seleção de índices. A heurística de seleção de índices considera os prováveis usos de índices de uma coluna, e combina esses a fim de formar índices de múltiplas colunas, verificando todas as possibilidades, deixando a cargo do Otimizador decidir se utiliza ou não os índices candidatos submetidos.

Em [23], é proposta uma ferramenta para seleção de índices para o IBM DB2. Este trabalho utiliza o próprio Otimizador do SGBD para enumerar os melhores índices para uma carga de trabalho e disponibiliza recursos de restrições de disco. Esse trabalho foi o primeiro que utilizou o próprio Otimizador para recomendar e avaliar os índices. Um aspecto constatado nesse trabalho, e em outros trabalhos existentes, é que se tratando de trabalhos direcionados a produtos comerciais, não há uma base de código publicamente disponível para se analisar a integração entre os módulos do sistema e outras questões de implementação.

O presente trabalho foi implementado no SGBD Objeto Relacional PostgreSQL, devido a ser um SGBD de código aberto *(Open Source)*, o que facilita a realização de análises e possíveis melhorias na arquitetura para seleção de índices desenvolvida.

CAPÍTULO 6

CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho se desenvolveu com o objetivo de prover mecanismos automáticos para a seleção de índices em bancos de dados relacionais. Inicialmente, foi feita uma revisão sobre conceitos relacionados ao processo de indexação e apresentada a metodologia utilizada em trabalhos que tratam do problema de seleção de índices. Estes trabalhos apresentam implementações de ferramentas de apoio ao DBA, na escolha de índices.

Foi abordada a complexidade do problema de seleção de índices em SGBDs relacionais, bem como um levantamento dos trabalhos sobre o assunto. Grande parte dos trabalhos descritos apresenta ferramentas de apoio ao DBA, totalmente separadas do SGBD, ocasionando uma discrepância entre o modelo de custos da ferramenta e o utilizado pelo Otimizador do SGBD.

Outra característica encontrada nas ferramentas, que dificulta a evolução das soluções, é o fato da maioria das soluções serem voltadas a SGBDs comerciais, impossibilitando a análise de seu código.

Neste trabalho, foi desenvolvida uma arquitetura para solução do problema de seleção de índices em bancos de dados relacionais e aplicado a solução ao SGBD de código aberto PostgreSQL, permitindo assim a validação e análise da solução. Um ponto importante a se destacar na arquitetura desenvolvida é a participação direta do Otimizador do SGBD na atividade de seleção de índices. O uso do Otimizador em outros trabalhos é empregado apenas para avaliar conjuntos de índices escolhidos pelas ferramentas.

Neste trabalho, foi explorado o uso do Otimizador na enumeração dos melhores índices para um comando SQL. Para isso, antes de iniciar o processo de otimização é empregada uma heurística

de escolha de índices candidatos, responsável por enumerar índices de uma e múltiplas colunas, para criação de índices hipotéticos no catálogo do SGBD. Após a criação dos índices hipotéticos, enumerados pela heurística, é iniciado o processo de otimização de consulta e fica, totalmente, a cargo do Otimizador selecionar os índices que trarão maior desempenho ao comando, sendo esses sugeridos ao DBA que decidirá criá-los ou não.

Para implementação da ferramenta de seleção de índices, inicialmente, foi alterado o SGBD, PostgreSQL, a fim de permitir a simulação de índices hipotéticos, necessários à arquitetura baseada em custo do Otimizador empregada e adicionado um módulo dentro do SGBD contendo o Algoritmo SIAIO, responsável pelas atividades envolvidas no processo de recomendação de índices.

As principais contribuições desta dissertação são as que seguem:

- desenvolvimento de uma arquitetura para seleção de índices, baseada em custos do Otimizador, com aplicação em um SGBD de código fonte aberto. A implementação da solução em um SGBD de código aberto, traz uma grande contribuição a comunidade de software livre e científica, promovendo a evolução do SGBD na área de ferramentas de auxilio ao DBA para esta plataforma e possibilitando a qualquer pesquisador analisar e contribuir com a pesquisa.
- desenvolvimento de uma heurística de enumeração de índices candidatos que tem por objetivo encontrar usos de colunas que podem ser indexadas. A principal vantagem da heurística é a enumeração de índices candidatos, explorando grande parte do espaço de busca de potenciais possibilidades, deixando a cargo do Otimizador a escolha dos melhores índices para um comando.

- avaliação de desempenho da solução, utilizando uma carga de trabalho do mundo real, através de um *benchmark* reconhecido pela comunidade científica. A viabilidade da solução foi atestada utilizando o *Toolkit* DBT-2, que mostrou as vantagens dos índices recomendados pelo Algoritmo SIAIO, em comparação aos índices sugeridos pelo *benchmark.*

Como perspectivas de trabalhos futuros sugere-se o que segue. Pode ser feita uma avaliação do comportamento da heurística em um Ambiente OLTP de grande porte, contendo tabelas com grande número de colunas a serem indexadas. Essa avaliação pode verificar possíveis gargalos na heurística de escolha de candidatos e sugerir soluções como métodos de corte, redução do número de candidatos, entre outras.

Outro trabalho, o qual pode ser estendida essa dissertação, seria a investigação de outras heurísticas para escolha de índices candidatos, utilizando métodos aleatórios como algoritmos genéticos.

O trabalho realizado nesta dissertação, ainda pode ser estendido, aplicando-se a arquitetura desenvolvida em outros tipos de sistemas, como por exemplo, Sistemas de Suporte a Decisão que utilizam um Data Warehouse. Neste caso, é provável que a arquitetura para seleção de índices desenvolvida ofereça vantagens, porque nesses sistemas consultas ao banco de dados são efetuadas com maior freqüência que inserções e atualizações, e a criação de índices pode ser realizada sem depender dos custos de atualizações dos índices, permitindo assim, obter melhor custo-benefício.

Por fim, o trabalho ainda pode ser aprimorado, adicionando e melhorando os cálculos estatísticos na recomendação de índices. É importante que sejam feitos aprimoramentos das estatísticas sobre índices hipotéticos, bem como a pesquisa de outras métricas de desempenho, para os índices recomendados. Por exemplo, o custo de atualização de índices. Para isso, é necessário um estudo detalhado sobre o modelo de custos do SGBD.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] AutoAdmin Project, *Database Group, Microsoft Research*, http://research.microsoft.com/en-us/projects/autoadmin.

[2] Barucci E. e Pinzani O. Optimal Selection of Secondary Indexes. *IEEE Transactions on Software Engineering*, 16(1):32-38, 1990.

[3] Capara, A., Fischetti, M. e Maio, D. Exact and Approximate Algorithms for the Index Selection Problem in Physical Database Design. *IEEE Transactions on Knowledge and Data Engeineering*, 7(6):955-967, 1995.

[4] Chaudhuri, S. e Narasayya, V. An Efficient, Cost-driven Index Selection Tool for Microsoft SQLServer. *Proceedings of the International Conference on very large Databases (VLDB'97)*, páginas 146-155, San Francisco, CA, EUA, 1997.

[5] Chaudhuri, S. e Narasayya, V. Microsoft Index Tuning Wizard for SQL Server 7.0. *Proceedings of the ACM SIGMOD International conference on Management of data*, páginas 553-554, Seattle, WA, EUA, 1998.

[6] Choenni, S., Blanken, H. e Chang, T. On the Selection of Secondary Indices in Relational Databases. *IEEE Data and Knowledge Engineering*, 11(3): 207-233, 1993.

[7] Comer, D. The Difficulty of Optimum Index Selection. *ACM Transactions on Data-base Systems (TODS)*. 3 (4): 440-445, 1978.

[8] Comer, D. The ubiquitous b-tree. *ACM Computing Surveys (CSUR)*, 11(2):121-137, 1979.

[9] Drake, J. D. e Worsley, J. C. Practical PostgreSQL. O'Reilly, 2002.

[10] Finkelstein, S., Schkolnick, M. e Tiberio, P. Physical database design for relational databases. *ACM Transactions on Database Systems (TODS)*, páginas 91-128, 1988.

[11] Frank, M., Omiecinski, E. e Navathe S. Adaptive and Automated Index Selection in RDBMS. *International Conference on Extending Database Technology (EDBT'92)*, páginas 277-292, Vienna, Austria, 1992.

[12] Generalized Search Tree (GiST). http://gist.cs.berkeley.edu.

[13] Gupta, H. , Rajaraman, A., e Ullman, J. D. Index Selection for Olap. *Proceedings of the IEEE International Conference on Data Engineering (ICDE'97),* páginas 208-219, Birmingham, Reino Unido, 1997.

[14] Harrison, Guy, Oracle SQL High-Performance Tuning , Prentice Hall, 1997.

[15] HORN P. *Autonomic Computing: IBM's Perspective on the State of Information Technology*, 2001. http://www.research.ibm.com/autonomic.

[16] IBM, DB2 Universal Database. http://www.ibm.com/db2.

[17] Kernighan B, Ritchie D. *The C Programming Language.* K&R. 1978.

[18] Kroenke, M. David. *Data Bases Foundation, Project and Implementation*, Prentice Hall, 1999.

[19] Korth, F., Silberchatz, A. e Sudarshan, S. *Sistemas de Bancos de Dados*. 3ª Edição. São Paulo: Makron Books. 1999.

[20] LANE, T. A Tour of PostgreSQL Internals. *Open Source Database conference, 2000.* http://www.postgresql.org/files/developer/tour.pdf.

[21] Lehman P., e Yao S. Efficient locking for concurrent operations on B+ trees. *Proceedings of the ACM Transactions on Database Systems,* 6(4):650-670, 1981.

[22] Levine, C. e DeWitt, D. Standard benchmarks for database systems. *ACM SIGMOD 1997 Industrial Session 5*. http://www.tpc.org/information/sessions/sigmod/indexc.htm.

[23] Lohman, G., Valentin, G., Zilio, D., Zuliani, M. e Skelley, A. DB2 advisor: An optimizer smart enough to recommend its own indexes. *Proceedings of the IEEE International Conference on Data Engineering- ICDE'00,* páginas 101-110, San Diego, CA, EUA.

[24] Maggie, Y. L. Ip, Saxton, L.V. e Raghavan V. On the Selection of an Optimal Set of Indexes. *IEEE Transactions on Software Engineering,* 9(2):135-143, 1983.

[25] MY SQL. http://www.mysql.com.

[26] Open Source Development Labs (OSDL). http://www.osdl.org.

[27] Oracle Corporation. http://www.oracle.com.

[28] PGADMIN3. http://www.pgadmin3.com.

[29] PostgrelSQL. http://www.postgresql.org.

[30] Postgresql global development group. http://developer.postgresql.org.

[31] Ramakrishnan R., Gehrke J. *Database Management Systems*, McGrawHill, 2003.

[32] Rozen, S. Shasha, D. A framework for automating physical database design. *Proceedings of the International Conference on Very Large Databases (VLDB).* páginas. 401-411, 1991.

[33] Salles, V. Marcos. *Criação Autônoma de Índices em Banco de dados*. Dissertação de Mestrado, Pontifícia Universidade Católica do Rio de Janeiro, 2004.

[34] Shasha, D. e Bonnet, P. *Database Tuning: Principles, Experiments and Troubleshooting Techniques*. Morgan Kaufmann, 2003.

[35] Sipser Michael. Introdução *À Teoria Da Computação*. Thomson Learning, 2ª Edição, 2007.

[36] SQL Server 2005. http://www.microsoft.com/sql.

[37] Zilio, Daniel C. *Physical Database Design Decision Algorithms and Concurrent Reorganization for Parallel Database Systems*, Tese de Doutorado, University of Toronto, 1998.

[38] Transaction processing performance council (TPC). http://www.tpc.org.

[39] Terada, R. *Desenvolvimento de algoritmos e estruturas de dados.* McGraw Hill, 1991.

[40] Ullman, Jeffrey D. *Principles of database and knowledge-base systems*, Computer Science Press, 1988.

[41] Weikum, G. Hasse, C. Monkeber, A. Zabback, P. The Comfort Automatic Tuning Project. *Informations Systems* 19(5): 381-432, 1994.

APENDICE A

ALGORITMO SIAIO NO POSTGRESQL

Nesta seção é apresentada a função principal do Algoritmo SIAIO, onde é possível verificar todas as chamadas de funções e todo o fluxo para seleção e recomendação de índices implementado.

```
// Variáveis globais
Cost gb_custo_atual;
Cost gb_custo_otm;
Cost gb_custo_inicia_atual;
Cost gb_custo_inicia_otm;

//============================================================================================
// Função principal do Algoritmo SIAIO.
// ===========================================================================================
List* siaio( Query*      query,                          // parâmetros do Otimizador
                         bool     isCursor,
                         int      cursorOptions,
                         ParamListInfo    boundParams,
                         Plan*    plan,                   // parâmetros utilizados pelo Alg. SIAIO
                         bool     retornoCandidatos)
{
        bool             gravaCandidatos = false;
        int              i;
        ListCell         *prev,                          // ponteiros p/ manipulação de lista
                         *cell,
                         *next;
        List*            opnos = NIL;
        List*            Candidatos = NIL;
        Cost             custoStartSemIndices;           // custo p/ iniciar comando SQL s/ índices
        Cost             custoTotalSemIndices;           // custo total sem índices
        Cost             novoCustoStart;                 // custo p/iniciar comando SQL c/ind. hip.
        Cost             novoCustoTotal;                 // custo total com índices hipotéticos
        Cost             custoStartSalvo;                // resultado de custoStart sem índices
        Cost             custoTotalSalvo;                // resultado de custoTotal sem índices
        FuncCandidatoList opnosResult;
        char *BTreeOps[] = { "<", ">", "<=", ">=","=", };

        custoStartSemIndices = plan->startup_cost;       // obtém custos sem índices hipotético
        custoTotalSemIndices = plan->total_cost;

        // cria lista contendo todos operados suportados por B-TREE
        for( i=0; i < lengthof(BTreeOps); ++i )
        {
                List* btreeop = list_make1( makeString( BTreeOps[i] ) );
                // Obtem os id's operador p/ operador, e armazena no vetor
                for(opnosResult = OpernameGetCandidatos( btreeop, '\0' );
                        opnosResult != NULL;
                        opnosResult = lnext(opnosResult) )
```

```
        {
                opnos = lappend_oid( opnos, opnosResult->oid );
        }
        pfree( linitial( btreeop ) );
        list_free( btreeop );
}
// inicio do esquadrinhamento da query
Candidatos = avalia_cmd_sql( query, opnos, NULL );
gb_query = query;

list_free( opnos );
if (list_length(Candidatos) == 0)
        goto DoneCleanly;

log_ic( "Candidatos Gerados", Candidatos );

// remove índices candidatos irrelevantes
Candidatos = apaga_ic_inuteis( Candidatos );
if (list_length(Candidatos) == 0)
        goto DoneCleanly;
log_ic( "Relevantes candidatos", Candidatos );

// cria os índices hipotéticos
Candidatos = implementa_ih( Candidatos );

// executa processo de otimização usando índices hipotéticos
plan = planner(query, isCursor, cursorOptions, boundParams);
novoCustoStart = plan->startup_cost;
novoCustoTotal = plan->total_cost;

// atribui valores globais aos custos de execução
gb_custo_atual = custoTotalSemIndices;
gb_custo_otm = novoCustoTotal;
gb_custo_inicia_atual = custoStartSemIndices;
gb_custo_inicia_otm = novoCustoStart;

// calcula os custos de execução
custoStartSalvo = custoStartSemIndices <= novoCustoStart ? 0 :
                custoStartSemIndices - novoCustoStart;

// verifica Se custo total sem índices é  menor que o novo custo
// SENÃO  recebe custo total sem indices - novoCustoTotal
custoTotalSalvo = custoTotalSemIndices <= novoCustoTotal ? 0 :
                custoTotalSemIndices   - novoCustoTotal;

// esquadrinha o plano de execução p/ encontrar índices usados
sinaliza_ic_planner( (Node*)plan, Candidatos );

// calcula o custo salvo para cada índice
{
        int4 Tamanho = 0;
        foreach( cell, Candidatos )
                Tamanho += ((IndiceCandidato*)lfirst( cell ))->pages;
```

```
                foreach( cell, Candidatos )
                {
                        IndiceCandidato *cand = ((IndiceCandidato*)lfirst( cell ));
                        cand->beneficio = (float4)custoTotalSalvo * (float4)cand->pages / Tamanho;
                        if (cand->beneficio > 1)
                        {
                                gravaCandidatos = true;
                        }
                }
        }
        // remove os índices hipotéticos
        apaga_ih( Candidatos );

        // remove índices candidatos não usados na Lista.
        for( prev = NULL, cell = list_head(Candidatos);
                cell != NULL;
                cell = next)
        {
                IndiceCandidato *cand = (IndiceCandidato*)lfirst(cell);
                next = lnext(cell);
                if (!cand->idcusado)
                {
                        pfree( cand );
                        Candidatos = list_delete_cell(Candidatos, cell, prev);
                }
                else
                        prev = cell;
        }
        if (list_length(Candidatos) == 0)
                goto DoneCleanly;

        // exibe candidatos utilizados constante no plano de execução
        log_ic( "Candidatos Usados", Candidatos );

        // grava as recomendações de índices em um arquivo
        if (gravaCandidatos)
        {
                grava_sugestoes(Candidatos);
        }

        // remove os índices candidatos da Lista
        if(!retornoCandidatos || !gravaCandidatos)
        {
                list_free( Candidatos );
                Candidatos = NIL;
        }
        DoneCleanly:

        // retorna a Lista de índices candidatos.
        return Candidatos;
}
```

APENDICE B

RESULTADOS BENCHMARK DBT-2

Nesta seção são apresentados resultados gerados pelo Toolkit DBT-2. Os dados são referentes à porcentagem de execução de cada uma das cinco transações do Benchmark DBT-2. O total de experimentos realizados foram 90. Como foi executado 10 vezes o mesmo experimento para cada cenário, o experimento escolhido para ser mostrado abaixo, foi aquele que teve um desempenho intermediário entre o pior e melhor resultado.

Cenário sem índices - escala 1 armazém

```
                                Response Time (s)
 Transaction      %    Average :    90th %        Total        Rollbacks      %
------------  -----  ----------------------  -----------  ---------------  -----
    Delivery   4.18      8.537 :    12.974          178                0   0.00
   New Order  45.61      7.364 :    11.802         1942               18   0.94
Order Status   3.73      0.805 :     1.564          159                0   0.00
     Payment  42.53      1.025 :     2.856         1811                0   0.00
 Stock Level   3.95      4.928 :     1.662          168                0   0.00
------------  -----  ----------------------  -----------  ---------------  -----
11.32 new-order transactions per minute (NOTPM)
180.2 minute duration
6 second(s) ramping up
```

Cenário com índices Toolkit DBT-2 - escala 1 armazém

```
                                Response Time (s)
 Transaction      %    Average :    90th %        Total        Rollbacks      %
------------  -----  ----------------------  -----------  ---------------  -----
    Delivery   3.99      0.089 :     0.162          207                0   0.00
   New Order  45.45      0.052 :     0.119         2356               19   0.81
Order Status   3.92      0.034 :     0.066          203                0   0.00
     Payment  42.28      0.017 :     0.031         2192                0   0.00
 Stock Level   4.36      0.003 :     0.003          226                0   0.00
------------  -----  ----------------------  -----------  ---------------  -----
12.67 new-order transactions per minute (NOTPM)
180.1 minute duration
8 second(s) ramping up
```

Cenário com índices algoritmo SIAIO - escala 1 armazém

```
                                Response Time (s)
 Transaction      %    Average :    90th %        Total        Rollbacks      %
------------  -----  ----------------------  -----------  ---------------  -----
    Delivery   3.80      0.055 :     0.061          195                0   0.00
   New Order  45.99      0.043 :     0.067         2358               12   0.51
Order Status   4.10      0.024 :     0.042          210                0   0.00
     Payment  42.13      0.015 :     0.013         2160                0   0.00
 Stock Level   3.98      0.004 :     0.006          204                0   0.00
------------  -----  ----------------------  -----------  ---------------  -----
12.81 new-order transactions per minute (NOTPM)
180.0 minute duration
3 second(s) ramping up
```

Cenário sem índices - escala 2 armazéns

```
                                Response Time (s)
 Transaction      %    Average :    90th %        Total        Rollbacks      %
------------  -----  ----------------------  -----------  ---------------  -----
    Delivery   4.33     53.792 :    77.466          205                0   0.00
   New Order  44.42     33.838 :    51.393         2104               28   1.35
Order Status   4.16      6.516 :    13.672          197                0   0.00
     Payment  43.45     13.850 :    33.153         2058                0   0.00
 Stock Level   3.65     39.199 :   226.584          173                0   0.00
------------  -----  ----------------------  -----------  ---------------  -----

10.57 new-order transactions per minute (NOTPM)
180.5 minute duration
0 total unknown errors
17 second(s) ramping up
```

Cenário com índices Toolkit DBT-2 - escala 2 armazéns

```
                                Response Time (s)
 Transaction      %    Average :    90th %        Total        Rollbacks      %
------------  -----  ----------------------  -----------  ---------------  -----
    Delivery   3.67      0.141 :     0.250          379                0   0.00
   New Order  45.17      0.077 :     0.133         4665               41   0.89
Order Status   3.87      0.067 :     0.094          400                0   0.00
     Payment  43.25      0.032 :     0.050         4466                0   0.00
 Stock Level   4.04      0.004 :     0.005          417                0   0.00
------------  -----  ----------------------  -----------  ---------------  -----

25.24 new-order transactions per minute (NOTPM)
180.1 minute duration
0 total unknown errors
14 second(s) ramping up
```

Cenário com índices algoritmo SIAIO - escala 2 armazéns

```
                                Response Time (s)
 Transaction      %    Average :    90th %        Total        Rollbacks      %
------------  -----  ----------------------  -----------  ---------------  -----
    Delivery   3.91      0.126 :     0.154          405                0   0.00
   New Order  45.66      0.105 :     0.138         4730               48   1.03
Order Status   3.93      0.087 :     0.118          407                0   0.00
     Payment  42.71      0.049 :     0.051         4424                0   0.00
 Stock Level   3.79      0.010 :     0.016          393                0   0.00
------------  -----  ----------------------  -----------  ---------------  -----

25.35 new-order transactions per minute (NOTPM)
180.1 minute duration
0 total unknown errors
17 second(s) ramping up
```

Cenário sem índices - escala 3 armazéns

```
                                Response Time (s)
 Transaction      %    Average :    90th %        Total        Rollbacks      %
------------  -----  ----------------------  -----------  ---------------  -----
    Delivery   4.32    115.204 :   154.769          151                0   0.00
   New Order  44.22     89.063 :   120.075         1546               12   0.78
Order Status   4.20     46.837 :    67.479          147                0   0.00
     Payment  43.56     51.379 :    77.578         1523                0   0.00
 Stock Level   3.69    108.193 :   432.463          129                0   0.00
------------  -----  ----------------------  -----------  ---------------  -----

8.35 new-order transactions per minute (NOTPM)
181.8 minute duration
0 total unknown errors
23 second(s) ramping up
```

Cenário com índices Toolkit DBT-2 - escala 3 armazéns

```
                                Response Time (s)
 Transaction      %    Average :    90th %        Total        Rollbacks      %
------------  -----  ----------------------  -----------  ---------------  -----
    Delivery   4.18      0.307 :     0.504          637                0   0.00
   New Order  45.71      0.221 :     0.422         6973               64   0.93
Order Status   4.11      0.176 :     0.365          627                0   0.00
     Payment  42.00      0.144 :     0.266         6407                0   0.00
 Stock Level   4.00      0.060 :     0.006          610                0   0.00
------------  -----  ----------------------  -----------  ---------------  -----

37.79 new-order transactions per minute (NOTPM)
180.1 minute duration
0 total unknown errors
25 second(s) ramping up
```

Cenário com índices algoritmo SIAIO - escala 3 armazéns

```
                                Response Time (s)
 Transaction      %    Average :    90th %        Total        Rollbacks      %
------------  -----  ----------------------  -----------  ---------------  -----
    Delivery   4.21      0.336 :     0.652          647                0   0.00
   New Order  45.41      0.219 :     0.451         6982               54   0.78
Order Status   3.47      0.243 :     0.585          533                0   0.00
     Payment  42.94      0.140 :     0.298         6602                0   0.00
 Stock Level   3.97      0.098 :     0.214          610                0   0.00
------------  -----  ----------------------  -----------  ---------------  -----

37.88 new-order transactions per minute (NOTPM)
180.1 minute duration
0 total unknown errors
18 second(s) ramping up
```