Wendel Góes Pedrozo

Tuning de Índices em Sistemas Gerenciadores de Banco de Dados Relacionais, utilizando Sistemas Classificadores

Curitiba

2019

Wendel Góes Pedrozo

Tuning de Índices em Sistemas Gerenciadores de Banco de Dados Relacionais, utilizando Sistemas Classificadores

Tese de doutorado apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Doutor em Informática.

Área de Concentração: Ciência da Computação

Linha de Pesquisa: Descoberta de Conhecimento e Aprendizagem de Máquina.

Orientador: Prof. Dr. Júlio Cesar Nievola. Coorientadora: Prof.ª Dr.ª Deborah Ribeiro Carvalho.

Curitiba

2019

Dados da Catalogação na Publicação
Pontifícia Universidade Católica do Paraná
Sistema Integrado de Bibliotecas – SIBI/PUCPR
Biblioteca Central
Pamela Travassos de Freitas – CRB 9/1960

Pedrozo, Wendel Góes
P372t *Tuning* de índices em sistemas gerenciadores de banco de dados
2019 relacionais, utilizando sistemas classificadores / Wendel Góes Pedrozo ; orientador: Júlio Cesar Nievola ; coorientadora: Deborah Ribeiro Carvalho. – 2019.
152 f. : il. ; 30 cm

Tese (doutorado) – Pontifícia Universidade Católica do Paraná, Curitiba, 2019
Bibliografia: f. 125-129

1. Aprendizado do computador. 2. Armazenamento de dados - custos. 3. Banco de dados - Gerência. 4. Banco de dados relacionais. 5. Desempenho. I. Nievola, Júlio Cesar. II. Carvalho, Deborah Ribeiro. III. Pontifícia Universidade Católica do Paraná. Pós-Graduação em Informática. IV. Título.

CDD 22. ed. – 005.756

Pontifícia Universidade Católica do Paraná
Escola Politécnica
Programa de Pós-Graduação em Informática

DECLARAÇÃO

Declaro para os devidos fins que o aluno **WENDEL GOES PEDROZO**, defendeu sua tese de doutorado intitulada **"*Tuning* de Índices em Sistemas Gerenciadores de Banco de Dados Relacionais, utilizando Sistemas Classificadores"**, na área de concentração Ciência da Computação, no dia 16 de maio de 2019, no qual foi aprovado.

Declaro ainda que foram feitas todas as alterações solicitadas pela Banca Examinadora, cumprindo todas as normas de formatação definidas pelo Programa.

Por ser verdade, firmo a presente declaração.

Curitiba, 14 de abril de 2020.

Prof. Dr. Emerson Cabrera Paraiso
Coordenador do Programa de Pós-Graduação em Informática
Pontifícia Universidade Católica do Paraná

Rua Imaculada Conceição,1155 Prado Velho CEP 80215-901 Curitiba Paraná Brasil
Fone: (41) 3271-1669 e-mail: secretaria@ppgia.pucpr.br

Dedico este trabalho à minha Família
em especial a meus pais
Benedito e Elma.

Agradecimentos

A Deus que me concedeu condições e forças para chegar até aqui, principalmente durante os anos de trabalho intenso no curso de doutorado.

Agradeço ao meu orientador Dr. Júlio Cesar Nievola, pela confiança na realização desta pesquisa, paciência, motivação e indicações sempre relevantes ao longo destes anos de orientação.

Agradeço a minha co-orientadora Drª. Deborah Carvalho Ribeiro, cuja sabedoria e conselhos enriqueceram por demais este trabalho.

Agradeço aos demais membros da banca Drª. Raquel Kolitski Stasiu, Dr. Marcos Didonet Del Fabro, Dr. Vidal Martins, Dr. Emerson Cabrera Paraiso e Dr. Alceu de Souza Britto Jr., por contribuírem significativamente para o aprimoramento desta pesquisa por meio de ideias, sugestões e críticas construtivas que me fizeram refletir sobre diversos aspectos.

Agradeço a Drª. Andreia Malucelli (ex-coordenadora) e ao Dr. Emerson Paraiso, responsáveis pela coordenação do PPGIa. Agradeço também a todos os professores do programa, que direta ou indiretamente contribuíram para minha formação.

Aos colegas do Grupo de Pesquisa de Descoberta do Conhecimento e Aprendizagem de Máquina, Alonso Decarli, Edenilson José da Silva, Erich Lacerda Malinowski, Elias Cesar Araújo de Carvalho, Leonardo Henrique Pereira, Luiz Henrique Giovanini Marrega, Rodolfo Botto de Barros Garcia, Zacarias Curi Filho e Patrícia Rucker de Bassi, pela colaboração, companherismo e compartilhamento de conhecimentos.

Aos demais amigos e colegas que fiz no PPGIa: André Brun, Bruno Souza, Carlos Eduardo (biluka), Cleverton Vicentini, Flávio de Almeida e Silva, Jean-Paul Barddal, Gregory Wanderley, Gustavo Bonacina, Irapuru Florido, Kelly Wiggers, Marcelo Zacharias, Marcia Pascutti, Patrícia Antoniolli, Rodrigo Siega, Diogo Olsen, Ronan Assumpção Silva, Jhonatan Geremias e Viviane Dal Molin.

Agradeço aos amigos da UTFPR e UFPR que contribuíram com sugestões e na revisão da tese e dos artigos frutos dessa pesquisa, além do suporte quando precisei me ausentar para conferências internacionais, em especial a Luzia Vidal, Fabiana Giacomini, Muriel Godoi, Fernando Barreto, Luciene Marin, Juliana Sgorlon, Lucio Rocha e Laio Seman.

Agradeço aos amigos da Cinq Technologies, Ivam Guimarães, Islenho Almeida e Rodrigo Amadi, que me auxiliaram algumas vezes na configuração do ambiente computacional.

Não seria possível a realização deste trabalho sem suporte financeiro. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001. Meus agradecimentos também a UTFPR, pela concessão do afastamento das atividades docentes.

Finalmente, um agradecimento especial a minha família, meus pais Benedito e Elma, a Claudia e aos meus filhos Gustavo e Giovana.

"Porque o Senhor Deus é sol e escudo;

o Senhor concede favor e honra;

não negará bem algum aos que andam na retidão."

Salmos 84:11

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Quadros

Lista de Abreviações

ACID: Atomicidade, Consistência, Isolamento e Durabilidade
AG: Algoritmo Genético
AM: Aprendizado de Máquina
Bit-map: Mapa de Bits
UCP: Unidade Central de Processamento
DBA: Administrador de banco de dados
DBGEN: *Database Generator*
DBMS: *Database management systems*
DBT3: *Database Test Suite 3*
DML: *Data Manipulation Language*
FTL: *Flash Translation Layer*
HDD: *Hard Disk Drive*
ITLCS: *Index Tuning with Learning Classifier Sytem*
LCS: *Learning Classifier System*
NoSQL: *Not Only SQL*
NP: *Nondeterministic Polinomial Time*
NV-RAM: *Non-Volatile Random Access Memory*
OLAP: *Online Analytical Processing*
OLTP: *Online Transaction Processing*
OSDL: *Open Source Development Lab*s
PDM: Processo de Decisão Markoviano
PG: Programação Genética
RAM: *Randon Acess Memory*
RL: *Reinforcement Learning*
RSL: Revisão Sistemática da Literatura
SATA: *Serial Advanced Technology Attachment*
SC: Sistemas Classificadores
SE: Sistemas Especialistas
SGBD: Sistema Gerenciador de Banco de Dados
SQL: *Structured Query Language*
SSD: *Solid State Drive*
SVM: *Support Vector Machine*
TPC: *Transaction Processing Performance Council*
XML: *Extensible Markup Language*

Lista de Símbolos

$\mathbf{B_{ej}}$	Benefício de uma estrutura ej
EC	Estimativa de custo de execução
BP	Benefício Percentual
$\mathbf{\|\|R\|\|_a}$	Número de registros atualizados por um comando SQL
$\mathbf{\|\|R\|\|_t}$	Número de registros da tabela
$\mathbf{\|\|R\|\|_p}$	Número de páginas da tabela
$\boldsymbol{c}$	Coeficiente percentual entre operação de E/S e custo de CPU
ECA	Estimativa de custos de atualização de registros de uma tabela
CCI	Custo de criação de um índice
AB	Abrangência
$\mathbf{NC_e}$	Número de comandos SQLs que um índice foi utilizado
$\mathbf{NC_{wt}}$	Número de comandos SQL da carga de trabalho
RC	Recompensa
PN	Punição
na	Comprimento da parte antecedente do classificador
nt	Número de símbolos # do antecedente do classificador
B	Aposta do classificador
k0	Coeficiente de aposta
k1	Coeficiente de participação da parte não referente à especificidade na aposta
k2	Coeficiente que correspondente à participação da especificidade na aposta
k3	Coeficiente que determina a importância da especificidade na aposta (padrão 1)
S	Especificidade do classificador
E	Energia (aptidão) do classificador
N	Vida média, medida em número de iterações
$\mathbf{c_s}$	Custo para buscar sequencialmente uma página no disco
$\mathbf{c_r}$	Custo para buscar aleatoriamente uma página no disco
$\mathbf{c_t}$	Custo de processamento de CPU de uma tupla
$\mathbf{c_i}$	Custo de processamento de CPU de uma entrada de índice
$\mathbf{c_o}$	Custo de processamento de CPU para realizar uma operação
CO	Custo de um operador O
$\mathbf{c_{sr}}$	Custo de leitura de uma página em disco de forma sequencial
$\mathbf{c_{sw}}$	Custo de escrita de uma página em disco de forma sequencial
$\mathbf{c_{rr}}$	Custo de leitura de uma página em disco de forma aleatória
$\mathbf{c_{rw}}$	Custo de escrita de uma página em disco de forma aleatória
Pdc	Percentual de melhora ou piora de desempenho
Pdt	Percentual de melhora ou piora do tempo

Resumo

Uma tarefa normalmente realizada por Administradores de Banco de Dados (DBAs) e ferramentas de auxílio ao DBA, para aumentar o desempenho de Sistemas Gerenciadores de Banco de Dados Relacionais (SGBDRs) é o ajuste de Projeto Físico. Nesse contexto o *tuning* de índices mostra-se relevante, uma vez que pode auxiliar significativamente na melhoria de desempenho do sistema de banco de dados. O emprego dessas ações, entretanto, deve considerar qual tipo de armazenamento está sendo utilizado pelo SGBDR, pois com o surgimento dos *Solid State Drives* (SSD) notou-se expressivas diferenças nos custos de E/S. A leitura e a gravação em SSDs são assimétricas com as operações de leitura muitas vezes mais rápidas que a escrita, além de apresentarem custos de E/S para acesso aleatório similar ao sequencial. Nesta tese é apresentado o algoritmo *Index Tuning with Learning Classifier System* (ITLCS), que combina aprendizado por reforço e algoritmos evolutivos para fornecer um mecanismo de ajuste de índice eficiente e flexível. O ITLCS foi desenvolvido especificamente para trabalhar em ambientes híbridos de armazenamento (HDD/SSD) e se baseia no modelo de desenvolvimento dos Sistemas Classificadores, utilizando um conjunto de regras no formato SE-ENTÃO, que além de representarem o conhecimento do sistema, são representações mais compreensíveis para pesquisadores e profissionais da área de banco de dados. Além disso, são empiricamente demonstradas as deficiências de SGBDRs na estimativa de custos de E/S em ambiente de armazenamento híbrido, e propostas alterações internas no modelo de custo do SGBDR utilizado, para que diferenças de custos entre dispositivos de armazenamento sejam reconhecidas. Experimentos realizados com o *benchmark* TPC-H, que simulam ambientes reais de sistemas de banco de dados, mostram que o desempenho do ITLCS foi superior ao das ferramentas EDB e POWA para os cenários de testes propostos, bem como demonstram a relevância das alterações propostas para o modelo de custos do SGBDR.

Palavras-chave: *tuning* de índices, aprendizagem de máquina, aprendizado por reforço, armazenamento híbrido, modelo de custos.

Abstract

A task typically performed by Database Administrators (DBAs) and advisor tools to increase the performance of Relational Database Management Systems (RDBMS) is the tuning of the Physical Design. In this context the tuning of indexes is relevant, since it can significantly help in improving the performance of the database system; these actions, however, should consider what type of storage is being used by the DBMS, since with the emergence of Solid State Drives (SSDs) significant differences in I/O costs were noticed. Reading and writing to an SSD is asymmetrical with read operations many times faster than writing, as well as having I/O costs for random access similar to sequential. In this thesis we present the algorithm Index Tuning with Learning Classifier (ITLCS), which combines reinforcement learning and evolutionary algorithms to provide an efficient and flexible index tuning mechanism. The ITLCS was developed specifically to work in hybrid storage environments (HDD/SSD) and uses as a development model the Learning Classification Systems (LCS), using a set of rules in the format IF-THEN, that besides representing the knowledge of the system, are more understandable representations for researchers and professionals in the database area. In addition, it is empirically demonstrated the deficiencies of RDBMS in the estimation of I/O costs on hybrid storage environment, and propose internal changes in the cost model of the RDBMS used, so that cost differences between storage devices are recognized. Experiments using the TPC-H benchmark, which simulate real-world database system environments, show that ITLCS performance was higher to that of the EDB and POWA tools for the proposed test scenarios, as well as demonstrating the relevance of the proposed changes to the RDBMS cost model.

Key-words: index tuning, machine learning, reinforcement learning, hybrid storage, cost model.

Capítulo 1

Introdução

A velocidade de processamento em CPU teve um aumento exponencial nas últimas décadas. Contudo, o número de entradas/saídas por segundo, em unidades de disco rígido, não acompanhou esse crescimento [1]. Em contraste com esse fato está o crescimento significativo no volume de dados a serem processados por sistemas de computação, o que demanda ainda mais desempenho dos dispositivos de armazenamento.

Neste contexto, o *tuning* de índices se evidencia relevante, uma vez que pode reduzir significativamente o número de acessos aos dispositivos de armazenamento. Índices correspondem a uma estrutura lógica e ordenada que mapeia os endereços onde os dados são armazenados fisicamente; seu uso é opcional, mas muito contribui para o aumento de desempenho [2]. Ações de *tuning* buscam diminuir o tempo de resposta e/ou aumentar o número de operações por unidade de tempo de determinada aplicação. Como exemplo de ações de *tuning* relacionadas à indexação temos: a seleção de quais tabelas/colunas devem conter índices, a reconstrução destes (caso estejam fragmentados), a migração e a sua exclusão.

Em Chaudhuri et al. [3] é demonstrado que, a depender do tamanho da instância do banco de dados (principalmente número de tabelas, número de colunas por tabela e tipos de índices utilizados), a seleção de índices é um problema de difícil tratamento e se enquadra em um problema NP *(Nondeterministic Polinomial Time)*. De fato, SGBDs podem conter milhares de tabelas, cada uma com centenas de colunas, logo milhares de possíveis índices candidatos devem ser considerados.

Outro ponto importante a se considerar em ações de *tuning* são os detalhes internos do SGBD utilizado, características dos dados e cargas de trabalhos [4] e características dos dispositivos de armazenamento secundário, pois os *Solid State Drive* (SSD) são ágeis para leitura [5], mas têm desempenho similar em escrita, se comparados aos *Hard Disk Drive* (HDD), além do custo monetário de um SSD ser maior.

Concernente aos dispositivos de armazenamento, devido às vantagens e desvantagens de ambos, de forma geral, corporações estão optando pelo armazenamento híbrido; ou seja, para aplicações com muitas operações de leitura escolhe-se os SSDs e para aplicações com características de muitas atualizações e inserções escolhe-se os HDDs.

Por meio desta tese busca-se desenvolver um algoritmo para *tuning* de índices, adaptável e flexível a ser aplicado em ambientes híbridos de armazenamento secundário.

1.1. Descrição do problema

SGBDs e ferramentas de auxílio ao DBA normalmente tratam os sistemas de armazenamento como uma "caixa-preta", não considerando as características particulares de E/S de cada dispositivo [6], tendo como consequência principal a obtenção de desempenho subótimo.

Devido ao uso heterogêneo de dispositivos de armazenamento, ambientes híbridos estão tornando-se cada vez mais comuns. Desse modo, são relevantes soluções de *tuning* que consigam diferenciar as características de E/S de cada dispositivo de armazenamento.

Existe na literatura iniciativas [15], [43] que propõem modificações internas nos SGBDs, como novos modelos de estimativas de custos sensíveis aos novos dispositivos de armazenamento. Contudo, a quantidade de estudos ainda são insuficientes para que essas abordagens sejam incorporados aos SGBDs atuais.

Verifica-se que ferramentas de auxílio ao DBA aplicáveis ao *tuning* de índices [4], [7], [8], [9] também não levam em conta as diferenças de custos de E/S entre diferentes tipos de dispositivos de armazenamento. Considerando que os custos predominantes em SGBDs são os de E/S dos dispositivos de armazenamento para a memória [2], então diferenciá-los se torna importante.

Entre as iniciativas já desenvolvidas para *tuning* de índices [4], [7], [8], [9], [10], [11], [18] verifica-se que muitas delas utilizam regras de *tuning* – também conhecidas como regras de ouro, em suas heurísticas internas [10],[11],[18]. No entanto, tais regras consideram

somente os custos de E/S dos HDDs, podendo não apresentar os resultados desejados, quando submetidos a dispositivos mais ágeis, como os SSDs.

Um exemplo de regra encontrado na literatura é sobre a criação de um índice *Bitmap* em uma coluna quando existe um número pequeno de valores únicos para a mesma [12]. Outro exemplo é encontrado em Chan e Ashdown [13], cuja proposta é sempre criar um índice para consultar uma tabela com mais de 50 blocos, filtrada por um predicado que obtenha menos de 15% de tuplas. Em outras palavras, a regra indica que o uso de um índice secundário será vantajoso se o retorno de tuplas for de até 15% da tabela; caso contrário, outras estratégias poderiam ser mais vantajosas, como por exemplo, varredura sequencial completa da tabela. De outra forma, em Lee et al. [14], é demonstrado que o uso de um índice em SSD foi vantajoso mesmo quando 30% das tuplas são recuperadas, evidenciando empiricamente que devido à velocidade de leitura em SSDs ser muito superior à dos HDDs, mesmo com o retorno de até 30% dos registros ainda é vantajoso percorrer um índice para recuperar as tuplas finais em vez de empregar outro método tal como varredura sequencial completa na tabela.

Embora existam várias iniciativas para resolver o problema em questão, existe uma lacuna no que tange a diferenciação dos custos de E/S entre HDD e SSD. Dado que os custos de E/S dos discos para a memória principal são os mais significativos nas operações típicas de um banco de dados [2] e que muitas corporações possuem ambientes de armazenamento secundário híbrido (SSD e HDD), então diferenciar os custos de E/S de cada dispositivo de armazenamento implica em um importante fator ao fazer o *tuning* de índices.

Além disso, as regras encontradas na literatura, ou mesmo recomendadas por DBAs e adotadas como base para construção de heurísticas de muitas ferramentas, consideram somente os custos de HDDs em sua concepção. No entanto, ao aplicar tais regras em ambientes com armazenamento SSD, suas ações poderão ter desempenho inferior ao seu potencial, e isso deve ser repensado.

Outro fator importante em soluções que utilizam regras para ações de *tuning* de índices, é que o modelo desenvolvido deve ser eficaz e compreensível o bastante para que os pesquisadores e profissionais da área consigam analisar e confiar nas ações sugeridas.

Com base no que foi apresentado, torna-se relevante buscar soluções automáticas para o problema de *tuning* de índices, com regras que sejam compreensíveis ao DBA e especializadas conforme o ambiente de armazenamento (HDD/SSD).

1.2. Hipótese de trabalho

A primeira hipótese defendida é a de que regras de tuning originalmente elaboradas para HDD devem ser modificadas, quando aplicadas em SSD.

A segunda hipótese é a de que, em ambientes híbridos de armazenamento (HDD/SSD), a alteração do modelo de estimativa de custos do SGBD, pode maximizar o desempenho do sistema.

Com estas hipóteses definidas, pretende-se buscar confirmá-las na *práxis*, por meio de um método experimental, definido no capitulo 5.

1.3. Objetivos

1.3.1. Objetivo geral

O objetivo geral desta tese é desenvolver um algoritmo baseado no modelo dos Sistemas Classificadores (SC) para realizar o *tuning* de índices em SGBDRs que utilizam armazenamento secundário híbrido (HDD e SSD).

1.3.2. Objetivos específicos

Para atingir o objetivo geral, os seguintes objetivos específicos foram traçados:

a) desenvolver heurísticas para recompensa e punição de classificadores;
b) analisar as diferenças entre regras especializadas em HDD e SSD, geradas pelo SC;
c) implementar o modelo de custos sensível à assimetria de leitura e escrita, proposto por Bausch et al. [15] adaptando-o ao SGBD PostgreSQL versão 10.1;
d) implementar formas para avaliar o algoritmo desenvolvido;
e) avaliar os resultados obtidos, comparando o algoritmo proposto com outras soluções para *tuning* de índices;
f) analisar as consultas do *benchmark*, identificando quais características têm maior benefício na utilização de um ou outro dispositivo de armazenamento.

Embora o objetivo geral desta tese se concentre na atividade de *tuning* de índices, vale lembrar, que existe a possibilidade do algoritmo proposto ser utilizado para outras ações de *tuning*. Devido ao modelo de SC ser uma abordagem genérica e flexível, pode-se utilizá-la em diferentes aplicações e domínios. Dessa forma, ações que envolvam outras estruturas que

compõem o projeto físico de banco de dados relacionais podem ser incluídas no SC com reduzidas adaptações.

1.4. Solução proposta

O algoritmo proposto, denominado *Index Tuning with Learning Classifier System* (ITLCS), apresenta uma solução baseada em regras para o problema de *tuning* de índices, utilizando como modelo de desenvolvimento os SCs. O ITLCS é o primeiro modelo baseado em SCs para essa finalidade, e foi projetado para ser adaptável a ambientes híbridos de armazenamento secundário (HDD e SSD).

Para realizar o *tuning* de índices, o SC cria e atualiza regras – chamadas de classificadores – na tentativa de cobrir a maior parte das situações de um ambiente real de banco de dados, ajustando o desempenho por meio da criação ou manutenção de índices. Além disso, o ITLCS identifica em qual tipo de dispositivo os dados estão armazenados e por meio da sua população de classificadores (alguns destinados a HDD e outros para SSD) são selecionados os classificadores que melhor possam otimizar o desempenho da consulta.

Outra característica do ITLCS é a capacidade de descoberta de novos classificadores por meio da meta-heurística evolutiva Algoritmo Genético (AG). Visando com isso gerar classificadores cada vez mais adaptados às características do ambiente do SGBDR.

Embora abordagens evolutivas tenham como desvantagem o tempo de convergência para obtenção da solução, neste trabalho isto foi significativamente reduzido. Graças ao emprego de índices hipotéticos[1] e pelo armazenamento de dados frequentemente utilizados, foram reduzidas as submissões repetidas a base de dados na extração de informações já calculadas anteriormente (informações que fazem parte da estrutura dos classificadores, como detalhada no capítulo 4, seção 4.2) utilizadas pelo AG.

O modelo proposto foi avaliado e comparado a outras soluções que também realizam o *tuning* de índices, como: *EnterpriseDB Index Advisor* [16] e *PostgreSQL Workload Analyzer* [17], em quatro cenários com diferentes configurações de armazenamento e de SGBDRs, obtendo resultados promissores que podem ser visualizados no capítulo 5.

[1] Índices hipotéticos não existem fisicamente em um SGBDR, mas são tratados pelo otimizador como se existissem fisicamente no SGBDR, pois assim como um índice fisicamente criado, este possui metadados e estatísticas armazenadas para fins de estimativas durante o processo de otimização de consultas. [75].

1.5. Estrutura do documento

Esta tese está organizada da seguinte forma: o Capítulo 2 apresenta a fundamentação teórica, cuja abordagem principal é *tuning* do projeto físico de bancos de dados e SCs. O Capítulo 3 descreve os trabalhos relacionados. O Capítulo 4 faz a abordagem metodológica, apresenta cada componente do algoritmo proposto, heurísticas desenvolvidas e modificações implementadas no modelo de custos do SGBD PostgreSQL. O Capítulo 5 apresenta o método de avaliação utilizado, os resultados obtidos e a discussão dos mesmos. O Capítulo 6 traz a conclusão, aponta as limitações do presente trabalho e sugere possibilidades de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta conceitos relacionados a *tuning* de projeto físico em SGBDs relacionais e conceitos sobre aprendizagem de máquina: suas técnicas, características e aplicações.

Primeiramente, a seção 2.1 introduz a classificação dos SGBDs e os conceitos envolvidos. Na seção 2.2 é abordado aspectos importantes de processamento de consultas. Em seguida, a seção 2.3 define as ações envolvendo projeto fisico de banco de dados. A seção 2.4 apresenta uma introdução a Indexação. A seção 2.5 trata as principais ações envolvendo *tuning* de banco de dados. A seção 2.6 discute o armazenamento secundário em SGBDs: tipos de armazenamento e diferenças. Por fim, na seção 2.7 introduzem-se os conceitos, técnicas e aplicações de aprendizagem de máquina que foram utilizados neste trabalho.

2.1. Sistemas Gerenciadores de Banco de Dados: conceitos e classificação

A arquitetura dos SGBDs evoluiu dos sistemas monolíticos, projetados em um único bloco formando um sistema fortemente integrado, para SGBDs modulares por projeto, utilizando por exemplo a arquitetura cliente/servidor.

Atualmente existem diversos tipos de SGBDs, com propósitos específicos, que podem ser utilizados para diversas aplicações. Em Elmasri e Navathe [12] é proposta uma classificação dos vários tipos de SGBDs por meio dos seguintes critérios: (i) modelo de dados

no qual o SGBD é baseado; (ii) número de usuários suportados pelo sistema; (iii) número de sites pelos quais o banco de dados está distribuído; (iv) o custo do SGBD; e (v) tipos de caminho de acesso (*access path*) para os arquivos armazenados.

Um dos principais modelos de dados utilizados nos SGBDs comerciais atualmente é o modelo relacional. O modelo de dados de objeto foi implementado em alguns sistemas comerciais, mas seu uso não foi muito difundido. Os SGBDs relacionais estão evoluindo continuamente e, em especial, têm incorporado muitos dos conceitos que foram desenvolvidos nos bancos de dados de objetos. Essa evolução gerou uma nova categoria de SGBDs chamada SGBDs objeto-relacional. Muitas aplicações legadas ainda utilizam os sistemas de banco de dados baseados nos modelos hierárquicos e de rede. O modelo XML (*eXtended Markup Language*), combina conceitos de banco de dados com os de modelos de representação de documentos, onde os dados são representados como elementos que podem ser aninhados em uma estrutura hierárquica. Nos últimos anos, SGBDs chamados NoSQL – acrônimo para *Not Only-SQL,* têm sido difundidos principalmente em aplicações para gerência e armazenamento de grandes volumes de dados semiestruturados ou não estruturados, utilizando modelos de dados orientado a colunas, orientado a documentos, Chave-valor e XML [20];

Nesta tese a solução foi direcionada para aplicações de missão crítica, com características OLAP. Devido a isso, optou-se pela utilização de SGBD relacional, que atende completamente as especificações ACID – acrônimo de atomicidade, consistência, isolamento e durabilidade.

2.2. Processamento de Consultas

Utilizando banco de dados relacionais, um conceito importante a entender é de qual forma as consultas submetidas são processadas. SGBDs relacionais permitem que os usuários ofereçam como entrada comandos em linguagem declarativa, geralmente a SQL, sem precisar se preocupar com a forma como os dados solicitados serão recuperados da unidade de armazenamento. Para isso, um mecanismo avançado é implementado pelo módulo de processamento de consultas. Metadados e outros métodos são combinados por meio de um plano de execução, ou plano físico.

Uma consulta em linguagem declarativa pode ser representada por vários planos de execução diferentes. No entanto, mesmo que dois planos sejam equivalentes – gerem o mesmo resultado, o tempo de execução e recursos (*hardware* e *software*) consumidos para

processamento podem variar de um para outro. A atividade de escolher qual o plano físico mais eficiente para determinada consulta é denominada planejamento [21]. As várias etapas percorridas, desde o momento do recebimento da consulta em linguagem declarativa até a recuperação dos dados são ilustradas na figura 2.1 e detalhadas nos próximos tópicos.

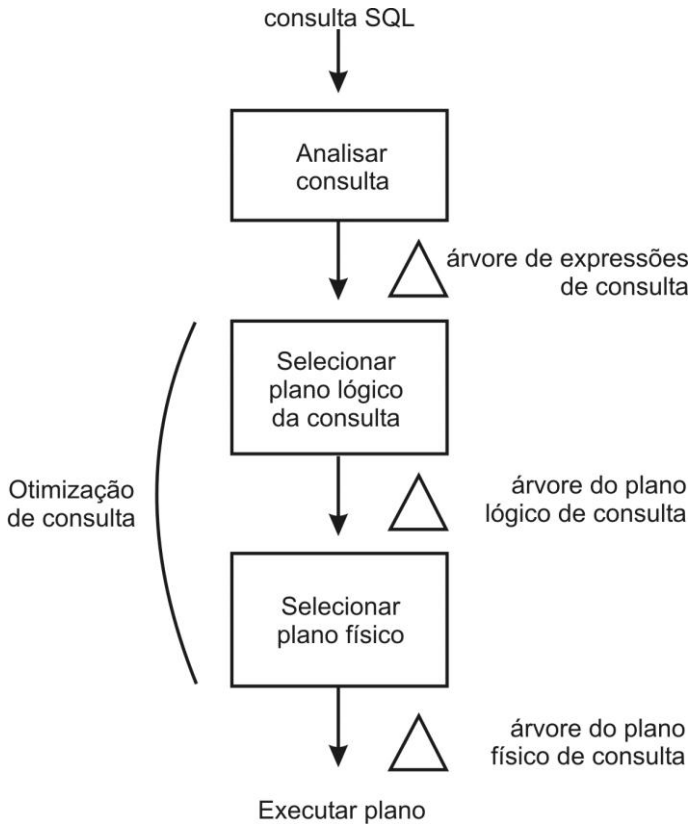


Figura 2.1. Etapas do processo de execução de uma consulta.
Fonte: Garcia-Molina et al.[21].

2.2.1. Análise

O objetivo da etapa de análise é transformar a consulta SQL em uma árvore de análise, conhecida como *parse tree*, de forma que seja possível sua manipulação pelo SGBD.

Outra atribuição desta etapa é a análise sintática da entrada recebida, que se dá em três etapas [21] [23]: (i) verificar se as relações presentes na cláusula SQL "FROM" existem no esquema de banco de dados; (ii) verificar se os atributos presentes na cláusula SQL "SELECT" ou "WHERE" existem nas relações referenciadas na cláusula "FROM"; e (iii) verificar se os atributos presentes na cláusula SQL estão utilizando os tipos adequados.

2.2.2. Reescrita

Nesta etapa, com a árvore de análise obtida na fase anterior, um plano de consulta algébrico é gerado, também chamado de plano lógico. Depois, esse plano é transformado em um plano equivalente, visando a diminuição de tempo e de recursos computacionais (memória, processamento e acessos a disco) na execução. Para obtenção do plano lógico, é necessária a substituição dos nós e das estruturas da árvore de análise por um ou mais operadores da álgebra relacional, visando produzir um plano de consulta mais eficiente [21].

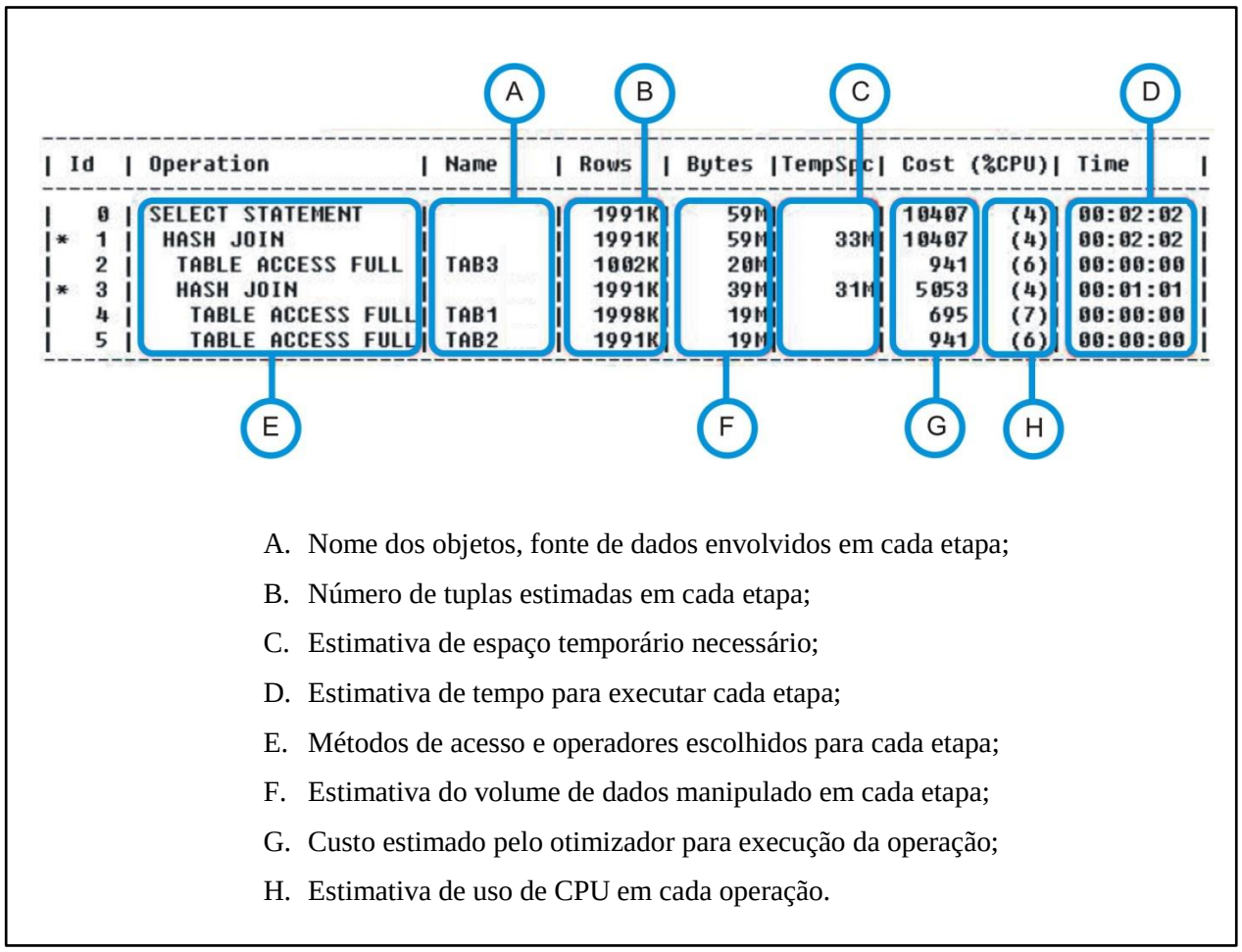
2.2.3. Planejamento

Como são analisados vários planos físicos, a avaliação de cada um é realizada individualmente considerando:

- sequência de operações comutativas e associativas como união, intersecção e junção;
- os operadores de junção utilizados para o processamento da consulta, como por exemplo: (i) baseados em *loops* aninhados (*nested loop join*); (ii) em ordenação (*sort merge join, merge join*) ou baseados em *hash* (*hash join*);
- o método de acesso aos dados contidos nas relações da consulta. Os métodos de acessos podem ser classificados em sequenciais (*seq-scan*) ou baseado em índices (*index-scan*);
- procedimento para repasse de argumentos entre operadores. Os resultados temporários podem ser armazenados, ou utilizam-se iteradores [21];
- a forma de repasse dos argumentos entre os operadores. O resultado pode ser armazenado temporariamente em disco (materialização), ou fazer uso de iteradores [21].

Para decidir qual plano físico tem maior eficiência é analisado o custo de cada plano. Esse custo é estimado pelo custo de cada método empregado, sendo obtido pelas estatísticas e pelo modelo de custos de cada SGBD. As estatísticas permitem obter dados e informações aproximadas sobre relações, como a seletividade e a cardinalidade. Já o modelo de custo possui vários fatores sobre os recursos computacionais de cada método de acesso, como: a quantia de memória principal necessária, a quantidade de acessos sequenciais/aleatórios em armazenamento secundário e tempo de utilização de CPU.

A figura 2.2 ilustra o plano de execução gerado por um SGBD relacional. Nele é possível verificar várias informações relevantes por meio do modelo de custos e das estatísticas.



A. Nome dos objetos, fonte de dados envolvidos em cada etapa;
B. Número de tuplas estimadas em cada etapa;
C. Estimativa de espaço temporário necessário;
D. Estimativa de tempo para executar cada etapa;
E. Métodos de acesso e operadores escolhidos para cada etapa;
F. Estimativa do volume de dados manipulado em cada etapa;
G. Custo estimado pelo otimizador para execução da operação;
H. Estimativa de uso de CPU em cada operação.

Figura 2.2 – Exemplo de plano de execução em um SGBDR.
Fonte: o autor.

2.2.4. Execução

Nesta fase, o plano físico selecionado na etapa anterior é interpretado e executado. Cada método descrito é invocado na ordem pré-determinada no plano físico. Neste momento são solicitados recursos de *hardware*, como: memória principal, processamento e requisições de escrita/leitura em dispositivos de armazenamento secundário. Como resultado desta etapa têm-se a consulta processada e o retorno de dados solicitados pela consulta.

2.2.5. Otimização baseada em custos e em heurísticas

Em um SGBD relacional o otimizador de consulta é o responsável por avaliar, entre várias alternativas disponíveis, o caminho de menor custo para execução de um comando com base nas estatísticas nele armazenadas incluindo: número de tuplas, média de tamanho de cada tupla, número de tuplas distintas em uma tabela, número de páginas físicas utilizadas para armazenar a tabela, entre outros.

As duas principais técnicas para implementação de otimizadores de consultas são baseadas em regras heurísticas e em estimativas de custos [12].

Na otimização baseada em heurísticas, uma consulta é modificada internamente para obter um melhor desempenho e normalmente a representação dessa consulta acontece por meio de um grafo ou uma árvore. Para a geração do plano de execução os predicados são processados na ordem que aparecem nos parâmetros da cláusula "*where*" do texto da consulta SQL e os métodos de acesso são escolhidos a partir de uma lista de regras definidas de acordo com o SGBD usado. Como normalmente é possível acessar os dados de diversas formas, para cada regra há um peso (*ranking*) ou prioridade. Quando houver mais de um método que possa ser utilizado, aquele com o menor peso ou a maior prioridade será o escolhido [12].

A otimização baseada em custos tem por objetivo comparar os custos da execução de consultas, entre várias estratégias de execução, para assim escolher aquela que tiver menor estimativa de custo [12]. A métrica de custo do otimizador é uma tentativa de calcular o custo computacional do processamento de um plano de execução. As funções estatísticas presentes no SGBD armazenam informações referentes ao número de tuplas, aos índices, à cardinalidade das tabelas, às chaves, à distribuição dos dados nas tabelas, ao tamanho das tabelas utilizadas, ao uso de CPU e os custos de E/S. Embora se busque utilizar a estratégia com menor custo, pode-se gerar um resultado que talvez não seja o de menor custo. Isto pode acontecer devido ao SGBD utilizar estatísticas desatualizadas. Assim, destaca-se a importância das atualizações periódicas. Contudo, não é razoável atualizar as estatísticas para cada alteração na base de dados em razão do alto custo para esta atividade [12].

2.3. Projeto Físico de Banco de Dados

O projeto físico tem como objetivos principais obter estruturas apropriadas para o armazenamento de dados garantindo um bom desempenho. Considerando um esquema conceitual, há muitas alternativas de projeto físico [12].

Um ponto importante é entender o perfil da carga de trabalho típica que o banco de dados deve suportar. Uma carga de trabalho consiste em consultas e atualizações a que o SGBD está sujeito. Os usuários também têm certos requisitos sobre o tempo máximo de espera para conclusão de determinadas consultas ou atualizações. Tais restrições de desempenho são determinantes para a escolha de quais atributos serão escolhidos para a construção de estruturas de acesso (índices, visões materializadas, etc.) [2].

Segundo Elmasri e Navathe [12], antes de conceber o projeto físico do banco de dados deve-se ter uma ideia do uso pretendido. Assim, tendo informações das consultas e transações esperadas, são analisados os seguintes fatores:

- as tabelas e os objetos que serão acessados pelas consultas;
- os atributos que são utilizados como condições de seleção para as consultas;
- os atributos que são utilizados nas condições de junção para ligar múltiplas tabelas ou objetos.

Ademais, deve-se mapear as tabelas e os atributos que serão frequentemente atualizados, pois para eles se deve evitar a implementação de estruturas de acesso como índices, uma vez que operações de inclusão, atualização e exclusão exigirão a atualização da estrutura de acesso. Se o benefício de uma estrutura de acesso ultrapassar o custo de atualização então a estrutura deverá ser criada [12].

2.4. Indexação

Um índice é uma organização de dados que permite acelerar o tempo de acesso às 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 frequentemente pelos leitores estão reunidos em uma relação alfabética, colocado ao final do livro. Este recurso permite que o leitor percorra o índice rapidamente buscando alguma palavra chave e vá direto para a página onde está o conteúdo de seu interesse. Assim como é tarefa do autor prever os itens que os leitores provavelmente vão procurar, é tarefa do DBA prever quais índices trarão benefícios.

Em SGBDs relacionais existem dois tipos básicos de índices: índices ordenados e índices *hash* [24]. O primeiro baseia-se em criar uma estrutura de índice para cada chave de pesquisa. Os valores das chaves de pesquisa são armazenados de forma ordenada, possibilitando o acesso rápido aos registros associados a cada chave de pesquisa. Já os índices *hash* baseiam-se em distribuir uniformemente os valores de chaves para uma determinada

faixa no disco de armazenamento, denominada *bucket* (balde). Para se obter um registro é aplicada uma função, denominada função de *hash* sobre a chave de pesquisa, e então é obtido o *bucket* que contém aquele registro [24].

2.4.1. Estrutura de dados para implementação de índices

Indexação baseada em árvore

A indexação baseada em árvore é uma das principais formas de se organizar registros. Esta técnica organiza as entradas de dados pelo valor da chave de pesquisa, e as pesquisas são então encaminhadas para as páginas corretas da entrada de dados. Entre as várias estruturas de árvore existentes, a árvore B+ é a mais amplamente utilizada, pois assegura que todos os caminhos da raiz até uma folha tenham o mesmo comprimento, ou seja, a estrutura está sempre balanceada em altura. Além disso, encontrar a página folha correta é mais rápido do que executar a pesquisa binária das páginas em um arquivo ordenado, porque cada nó não folha pode acomodar um número muito grande de ponteiros para os nós, e a altura da árvore raramente é maior que três ou quatro na prática.

No exemplo da árvore B+ da figura 2.3, a altura da árvore é de nível três. Todas as pesquisas iniciam no nó superior, chamado raiz, sendo direcionadas para o nível inferior da árvore, chamado de nível folha, que contêm as entradas de dados, que neste exemplo são os registros de funcionários. O número de E/S para recuperar uma página folha desejada é quatro, incluindo a raiz e a página folha.

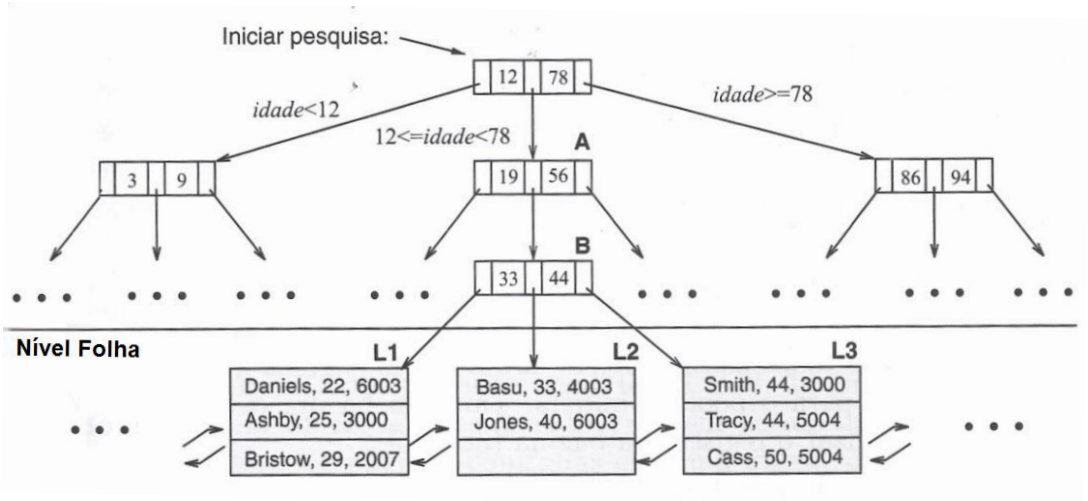


Figura 2.3 – Índice estruturado como árvore B+.
Fonte: extraído de Ramakrishnan e Gehrke [2].

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 podem ser observados principalmente na possibilidade de indexar milhões de chaves mantendo a altura três. 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.

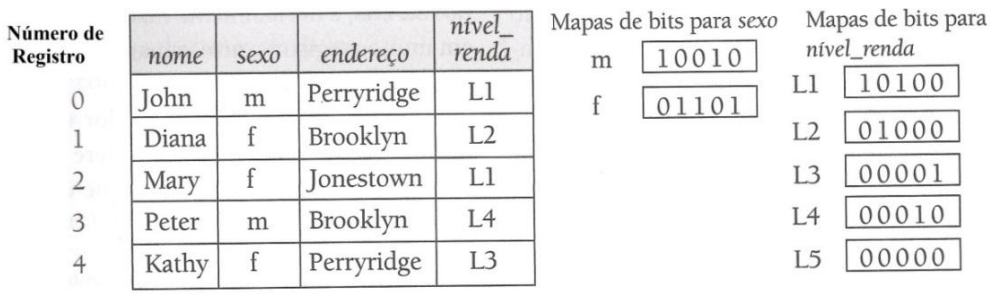
Índices de mapa de bits

Dependendo da seletividade de um atributo de uma relação, uma opção para indexação é a utilização de índices de mapa de bits (*bitmap*) que são indicados quando a seletividade de um atributo é menor que 50% [13].

Segundo Silberchatz et al. [24] um mapa de bits é simplesmente um *array* de bits. Desta forma, um índice de mapa de bits sobre o atributo A da relação r consiste em um mapa de bits para cada valor que A pode assumir.

A figura 2.4 ilustra a relação info_cliente com o atributo sexo, que pode assumir apenas os valores m (masculino) ou f (feminino), onde o i-ésimo bit para m é definido como 1 se o valor de sexo do registro numerado com i for m e outros bits do mapa de bits para m são definidos como 0. De forma equivalente, para f o mapa de bits tem o valor 1 para os bits com o valor f para o atributo sexo e os outros bits constam como 0.

Já o atributo nível_renda, pode assumir 5 valores. Embora os dados de nível_renda possa ter mais de 5 valores, para simplificar o analista de dados preferiu dividir em faixas de valores, possibilitando assim a utilização de um índice de mapa de bits. Desta forma as faixas de renda foram divididas da seguinte forma: L1: 0-9999, L2: 10.000-19.999, L3: 20.000-39.999, L4:40.000-69.999, L5: 70.000-infinito.



Número de Registro	*nome*	*sexo*	*endereço*	*nível_ renda*
0	John	m	Perryridge	L1
1	Diana	f	Brooklyn	L2
2	Mary	f	Jonestown	L1
3	Peter	m	Brooklyn	L4
4	Kathy	f	Perryridge	L3

Mapas de bits para *sexo*

m	10010
f	01101

Mapas de bits para *nível_renda*

L1	10100
L2	01000
L3	00001
L4	00010
L5	00000

Figura 2.4. Índice de Mapa de Bits para atributo sexo e nível_renda.
Fonte: Extraído de Silberchatz et al. [24].

Definição sobre qual tipo de índice criar

Uma recorrente questão na atividade de seleção de índices secundários é qual tipo de índice criar. Embora alguns SGBDs disponibilizem muitas opções diferentes, normalmente os tipos de índices mais utilizados e que a maioria dos SGBDs suportam são índices estruturados em árvore B, organização *hash* e mapa de bits.

Segundo Silberchatz et al. [24], se as consultas de intervalo forem comuns, os índices de árvore B são preferíveis ao invés dos índices em *hash*. No entanto, os índices estruturados em *hash* funcionam muito bem quando são utilizados com operador de igualdade, principalmente durante as junções de relações, para encontrar um ou mais registros.

Os índices de mapa de bits (*bit-map*) dispõem de uma representação bastante compacta para indexar atributos com baixa seletividade, ou seja, valores muito pouco distintos [24]. Desta forma, operações de intersecção são extremamente rápidas nos índices de mapas de bits e o otimizador do SGBD certamente optará por um índice mapa de bits ao invés de realizar uma varredura completa na tabela.

2.4.2. Seleção de índices

Índices podem acelerar o desempenho de várias operações em um SGBD como: operações de seleções, ordenações e junções [12].

Embora a criação de índices contribua em muitos casos para a melhoria de desempenho, a sua criação de forma equivocada pode reverter em desvantagens [68]:

- índices que nunca são utilizados e que obrigatoriamente necessitem ser atualizados a cada alteração sobre a tabela indexada;
- junções de várias tabelas que demoram horas devido à existência de índices fragmentados.

Portanto, a atividade de selecionar índices que realmente sejam úteis é um desafio para o DBA. Um índice pode ser considerado útil quando reduz o tempo de execução de comandos SQL, presente em uma carga de trabalho, sem degradar significativamente o desempenho de outros comandos.

Na literatura é possível encontrar autores, como Shasha e Bonnet. [68] e Bruno [52], que descrevem boas práticas relacionadas à sintonização de índices, conforme segue abaixo:

- utilizar com parcimônia índices sobre colunas frequentemente atualizadas. Pela necessidade de atualização do índice a cada vez que ocorre uma alteração sobre a

tabela subjacente, deve-se restringir a utilização em tabelas com características de atualizações frequentes;

- utilizar índices em chaves primárias e chaves estrangeiras em operações de junções complexas. Em operações de junções, os índices podem auxiliar na diminuição do volume de dados intermediários para realizar operações de junções;
- colunas presentes em cláusulas "*where*" de uma consulta são potencialmente candidatas à indexação. Predicados de igualdade ou por faixas de valores/intervalos podem ser eficientemente resolvidos se índices estiverem presentes nas colunas propriamente ditas;
- limitar o uso de índice em tabelas pequenas. Em tabelas que ocupem menos que três páginas, a busca utilizando índices será preterida e métodos como varredura completa normalmente terão um custo menor. Deve-se considerar também o custo para gerenciar e atualizar essas estruturas.

Complexidade computacional em seleção de índice

Na teoria de algoritmos, problemas tratáveis - ou seja, solúveis 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).*

Atualmente, na classe de Problemas NP é onde reside a maioria dos problemas computacionais e de difícil tratamento, pelo fato do número de computações do melhor algoritmo conhecido crescer exponencialmente em função do tamanho da instância.

Já foi demonstrado por Chaudhuri et al. [3] e Comer [60], que dependendo do tamanho da instância o problema de seleção de índices se enquadra em uma subclasse dos problemas NP, denominado NP-Difícil. De fato, a operação de encontrar o conjunto de índices adequado a uma carga de trabalho não pode ser negligenciada.

Segundo Ramakrishnan e Gehrke [2], o número total de diferentes índices que é possível sobre uma tabela pode ser conhecido utilizando a expressão 2.1 a seguir:

$$\sum_{k=1}^{n} \frac{n!}{(n-k)!} \qquad \text{(Exp. 2.1)}$$

Onde

n: é o número de colunas da tabela;

k: é o número de colunas que um índice usa;

Para exemplificar, considere que se tem interesse em 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)!.

Aplicando a expressão 2.1 em uma tabela com 10 colunas (n=10) a serem indexadas, contendo índices de 1, 2 e 5 colunas (k=1; k=2; e k=5; respectivamente), tem-se 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 OLAP e OLTP atuais, encontrar o conjunto ótimo de índices adequado é um desafio combinatório.

2.4.3. Manutenção de índices

Um problema importante a considerar acerca de índices é o processo de sua manutenção. A estrutura de dados conhecida como árvore B+ é sem dúvida a estrutura mais amplamente utilizada para implementação de índices em banco de dados relacionais, por ser uma estrutura que se adapta bem a mudança e suporta tanto consultas por igualdade quanto por intervalos [2]. No entanto, devido as suas características estruturais, um grande número de comandos SQL de *insert*, *update* ou *delete*, fatalmente provocará divisões de páginas no nível folha do índice, processo conhecido como fragmentação *(page-splits).* As figuras 2.5(a) e 2.5(b) demonstram a ordem em que o índice é percorrido em uma consulta.

A figura 2.5(a) apresenta um índice íntegro, sem fragmentação. Já a figura 2.5(b) retrata o índice após a realização de uma inserção de dados, que ocasionou uma divisão de página. Como não havia espaço no índice para inserção da chave WAC, a página foi dividida em duas e a inserção foi realizada. Deve ser observado na figura 2.5(b) que para percorrer o índice serão necessários mais deslocamentos do que no índice da figura 2.5(a), e isso é um problema em operações de varreduras porque exigirá um esforço maior. Uma possível solução para o problema de divisão de páginas seria a reconstrução do índice, processo que deixará

novamente o índice ordenado fisicamente, provendo maior desempenho em operações de varredura [71]. No entanto, a reconstrução do índice pode ser onerosa para o banco de dados, dependendo do tamanho do índice.

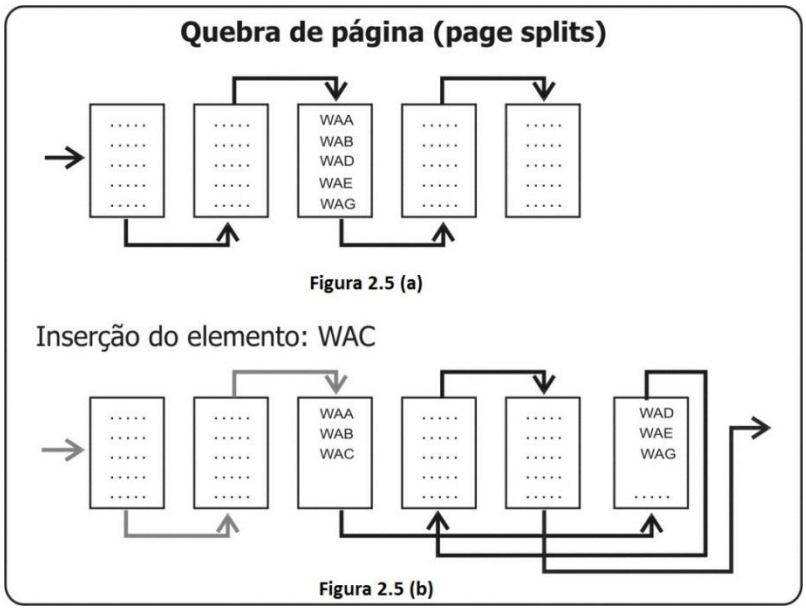


Figura 2.5 – Processo de fragmentação de um índice.
Fonte: o autor.

2.5. *Tuning* de banco de dados

Durante o início de um projeto de banco de dados, informações precisas sobre carga de trabalho podem ser difíceis de obter. Desta forma, é importante ajustar um banco de dados depois de ter sido concluído o projeto físico. A distinção entre projeto de banco de dados e *tuning* é um tanto arbitrária. Pode-se considerar o processo de projeto terminado quando o esquema conceitual inicial estiver projetado e um conjunto de decisões de indexação for concluído. Neste caso, quaisquer alterações posteriores no esquema conceitual ou no projeto físico são consideradas como *tuning* [2].

Outro conceito a ser observado é a diferenciação de *tuning* de banco de dados e otimização de consultas, apesar da relação entre essas áreas. A otimização de consultas é o processo realizado pelo SGBD que busca selecionar o plano de execução mais eficiente para uma consulta. Portanto, é um processo automático executado pelo próprio SGBD. Já o *tuning* de banco de dados é realizado normalmente pelo DBA e refere-se à atividade de ajustar os parâmetros, as configurações e as estruturas de um sistema de bancos de dados visando a maior rapidez e eficácia em suas ações. Para realizar essa atividade alguns fatores são determinantes. São eles:

a) carga de trabalho: conjunto de tarefas (consultas ou atualizações) realizadas por unidade de tempo;

b) *throughput*: vazão das transações (requisições realizadas por unidade de tempo);

c) recursos: *hardware* disponível e as ferramentas de *softwares* disponíveis.

Com isto, pode-se definir *tuning* como uma otimização nos recursos usados para aumentar o *throughput*, visando o aumento da capacidade de executar uma carga de trabalho no menor tempo possível.

Melhorias em *hardware* de processamento e/ou discos mais rápidos surgem como uma boa opção para incremento consistente e imediato no desempenho. Entretanto, pode-se optar pelo ajuste de outros recursos, como a realocação de memória e até no próprio sistema operacional utilizado. Tais parâmetros podem afetar o desempenho dos discos rígidos, número de arquivos que o sistema poderá manter em aberto ao mesmo tempo, *threads*, ambiente gráfico, etc.

Desta forma, um DBA deve estar atento não somente ao SGBD, mas a todo o ambiente que o compõe, visto que tratar de um problema isoladamente é uma falha que poderá não ter solução se todas as partes não forem avaliadas e balanceadas.

Os SGBDs atuais possuem certo grau de complexidade em razão das centenas de parâmetros que podem ser ajustados. Esses ajustes exigem dos DBAs conhecimento aprofundado dos algoritmos internos utilizados pelo SGBD, bem como das inter-relações entre os ajustes de parâmetros. Desta maneira, o nível de conhecimento dos profissionais especializados na realização de *tuning* de banco de dados é cada vez maior, sendo necessária uma quantidade crescente desses profissionais para suprir as necessidades atuais.

Todos esses fatores fizeram com que o meio acadêmico e a indústria adotassem como objeto de pesquisa a realização do *tuning* de banco de dados de forma automática. Este conceito visa auxiliar o DBA na complexa missão de administração dos SGBDs, visando manter estruturas de acesso, como índices, visões materializadas e partições de tabelas sempre adequadas às cargas de trabalho, visando alto desempenho global do SGBD.

2.6. Armazenamento secundário em SGBDs

Por décadas os HDDs têm sido a solução dominante para o armazenamento secundário em SGBDs, sendo considerados a melhor relação custo-benefício, mesmo tendo menor

desempenho em relação a outras tecnologias de armazenamento não volátil, como por exemplo, as memórias semicondutoras NV-RAM [65]. Os HDDs têm desempenho melhor em acessos sequenciais – pois favorecem dados que estão localmente próximos – do que em acessos aleatórios, devido à necessidade de movimentação de partes mecânicas para tal.

Com o advento dos discos em estado sólido (SSDs), baseados em memória *flash*, ocorreu uma revolução nas tecnologias de armazenamento. Como corretamente previsto em Gray e Fitzgerald [58], *"tape is dead, disk is tape and flash is disk"*. *Flash* SSDs devido a serem dispositivos sem peças mecânicas, compostos basicamente de semicondutores (chips), apresentam baixas taxas de consumo de energia, resistência a choques e menor tamanho em relação aos HDDs.

Atualmente, com ambientes de armazenamento cada vez mais híbridos, devido à utilização heterogênea de dispositivos de armazenamento nos SGBDs, este cenário se torna ainda mais complexo. Na literatura é possível verificar que na última década razoável esforço foi despendido por pesquisadores em busca de adaptação e novas técnicas de acesso a dados em SGBDs que utilizam armazenamento em SSDs. Como meio de identificar, avaliar e interpretar trabalhos disponíveis relevantes sobre os impactos da utilização de SSDs em SGBDs, foi realizado no contexto desta tese uma revisão sistemática da literatura (RSL) sobre o tema, disponível no Apêndice A, onde foram identificando 1974 artigos de cinco bases de dados que após eliminação de títulos duplicados e da aplicação de critérios de exclusão, resultaram em 25 artigos, claramente de acordo com os objetivos da revisão sistemática, sendo então analisados e classificados.

2.6.1. *Solid State Drive*

A memória Nand *flash* é o bloco base de construção de SSDs. Um chip *flash* compõe-se de uma cadeia de células *flash*, que podem armazenar bits 0 ou 1. Um bloco *flash* é subdividido em várias páginas [61].

São três ações que podem ser executadas em um dispositivo *flash*: ler, programar/escrever e apagar. Uma operação de leitura pode ocorrer aleatoriamente em qualquer lugar de uma página. Uma operação de programação/escrita pode ser executada somente em um bloco limpo. Uma operação de apagamento por sua vez, tem a funcionalidade de atribuir 1 para todos os bits dentro de um bloco, sendo endereçáveis somente por bloco.

Algumas características dos dispositivos de memória *flash* são únicas e estão resumidas a seguir:

- No Update: ao contrário dos HDDs, não é possível sobrescrever setores em uma memória *flash*. Para substituir os dados existentes, primeiro um bloco inteiro deve ser limpo e somente após isso os novos dados podem ser escritos. Este processo é conhecido como *erase-before-write,* o que significa que todos os outros setores do bloco precisam ser lidos e carregados para a memória principal e, em seguida, escritos no disco novamente [74];
- Ausência de latência mecânica: por ser um dispositivo puramente eletrônico, como reportado na tabela 2.1, operações de leituras podem ser mais de 300 vezes mais rápidas [5]. Isso ocorre porque na escrita gasta-se mais tempo para energizar uma célula do que simplesmente ler o seu estado, como ocorre nas operações de leitura.
- Limitado número de gravações: normalmente variando entre 10.000 a 100.000 ciclos por bloco. Após exceder esse limite, o bloco torna-se não confiável. Para evitar que alguns blocos se tornem inutilizáveis muito antes de outros no mesmo dispositivo, são empregadas técnicas que distribuem o nível de desgaste uniformemente em todo segmento de memória [74];
- Paralelismo: pode ser obtido de duas formas: (i) em nível de canal podendo ser operados independente e simultaneamente e (ii) em nível de pacotes *flash* ligados ao mesmo canal. Além disso, quase todos os SSDs modernos suportam mecanismos *Native Command Queuing* (NCQ) que permitem que SSDs aceitem múltiplas requisições de E/S em paralelo [43].

Tabela 2.1 Medidas de tempo de acesso para HDD e SSD.

	Tempo de acesso		
	Leitura	Escrita	Apagar (e*rase)*
HDD	20,4 ms (4 KB)	21,5 ms	-
Flash SSD	0,06 ms (4 KB)	0,5 ms (4 KB)	1.5 ms (256 KB)

Fonte: Adaptado de Wang e Wang [5].

Devido às novas características da memória *flash*, e principalmente ao número limitado de vezes que os blocos *Flash* podem ser programados e apagados de forma confiável, um componente essencial é a *Flash Translation Layer* (FTL). Trata-se de uma camada de *software* que fornece uma interface de dispositivo de bloco para camadas superiores por meio

de operações que emulam uma unidade de disco, e é responsável por funções como: nivelamento de desgaste, tradução de endereços e controle de informações de mapeamento entre as páginas lógicas e as páginas físicas.

Para melhor visualizar as diferenças, vantagens e desvantagens entre HDD e SSD, no quadro 2.1 apresenta-se um comparativo entre custo, vida útil aproximada, taxa de transferência, falha, consumo de energia e latência de acesso.

Quadro 2.1. Comparativo entre HDD e SSD.

Característica	HDD	SSD
Custo em U$ (dólar) por *Gigabyte*	aproximadamente U$ 0,06 por *gigabyte*, (considerando HDD de 4TB)	aproximadamente U$ 0,10 por *gigabyte* (considerando SSD de 1 TB)
Vida Útil (resistência)	4 anos (20 GB por dia)	3 anos (20 GB por dia)
Taxa de transferência	300 MB/S	66 MB/S
Taxa de Falha	2 (milhões de horas)	1,5 (milhões de horas)
Consumo médio de energia (*Watts*)	10W	1W
Latência de Acesso	12 ms	0,1 ms

Fonte: extraído de Graefe [38], Baxter [79].

2.7. Aprendizado de Máquina

Aprendizado de máquina (AM) é uma subdivisão da área de inteligência artificial cujo objetivo é, a partir de experiência acumulada, desenvolver técnicas e métodos computacionais que permitam a computadores a tomada de decisão [34].

Na figura 2.6 apresenta-se a hierarquia em que se encontram as principais áreas e subáreas que o presente trabalho atua, com destaque à Aprendizagem de Máquina, Computação Evolucionária e Sistemas Classificadores.

As técnicas de AM permitem a obtenção de conhecimento automático por meio de um conjunto de dados. Entre as pesquisas na área de AM destaca-se o aprendizado indutivo, que visa produzir (induzir) modelos automaticamente a partir de dados, como as regras e os padrões. A indução é caracterizada por fazer inferências baseadas em um subconjunto particular de dados [28], e pode ser dividido em duas categorias, conforme segue abaixo.

- Aprendizado supervisionado: utilizam algoritmos de indução que a partir de dados rotulados, realizam inferências. Alguns exemplos são os algoritmos baseados em Árvores de Decisão, Redes Neurais Artificiais e as Máquinas de Suporte Vetoriais. (*SVM - Support vector machine*).
- Aprendizado não supervisionado: para o algoritmo são disponibilizados apenas os exemplos de entrada, não existindo informação sobre a saída esperada. Uma das

subáreas principais que utilizam aprendizado não supervisionado é agrupamento de dados, como o algoritmo *k-means* [34].

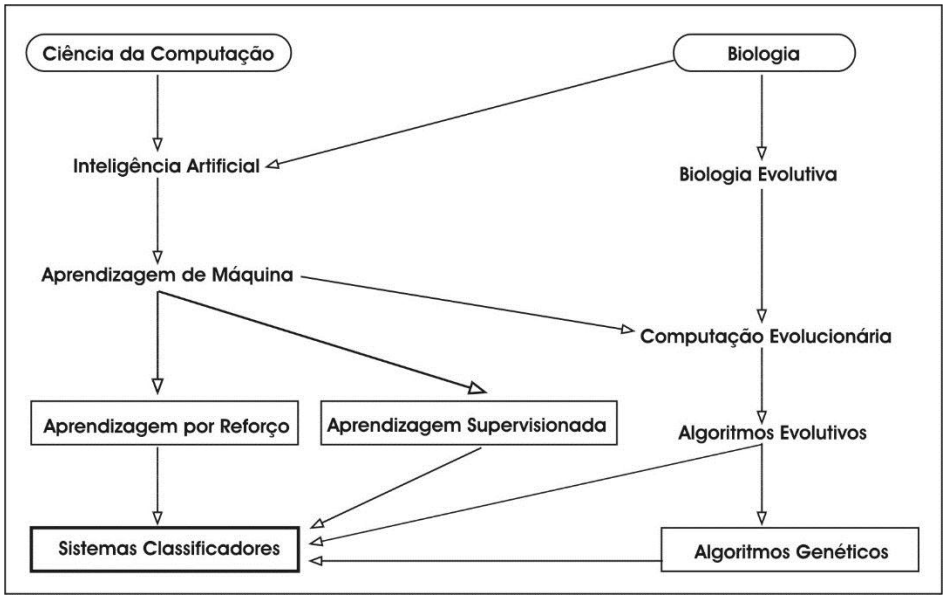


Figura 2.6. Hierarquias envolvendo Sistemas Classificadores.
Fonte: adaptado de Urbanowicz e Moore [19].

Outra categoria de aprendizado na área de AM é o aprendizado por reforço, na qual o algoritmo desenvolve um método de como comportar-se com base nas respostas as suas ações anteriores [59].

Nesta tese optou-se pela utilização da aprendizagem por reforço, principalmente devido às caracteristicas do problema de *tuning* de projeto físico de índice, no qual o SC proposto ajusta o desempenho de um SGBD por meio de ações de criação/manutenção de índices e da observação dos resultados dessas ações, sendo este comportamento, de forma geral, similar à atuação de um DBA, que normalmente toma decisões com base em experiências já realizadas. No problema tratado nesta tese, considerando uma entrada de dados (consulta SQL) o sistema realizará ações para ajuste de projeto físico (índices) baseado em um conjunto de regras de classificação que são compreensíveis, e podem ser facilmente analisadas por profissionais de banco de dados.

2.7.1. Aprendizagem por reforço

Mitchell [70] define que o aprendizado por reforço – AR, em inglês *Reinforcement Learing*, tem por objetivo o estudo de como um agente autônomo, que tem conhecimento do seu ambiente, pode aprender a decidir quais as melhores ações a serem tomadas. A utilização

de AR é recomendada quando não se dispõe de modelos a priori ou quando não se consegue obter exemplos apropriados das situações que o agente irá enfrentar [86].

Em AR, um agente aprendiz é aquele que, a partir da interação com o ambiente que o cerca, aprende de maneira autônoma uma política ótima de atuação e interage com o ambiente em intervalos de tempos discretos em um ciclo de percepção-ação, como ilustrado na figura 2.7. Na figura o agente aprendiz observa, a cada passo de interação, o estado corrente *st* do ambiente e escolhe a ação *at* para realizar. Ao executar a ação *at*, que possibilita alterar o estado do ambiente, o agente recebe um sinal escalar de reforco $r_{s,a}$ (penalização ou recompensa), que indica quão desejável é o estado resultante s_{t+1}.

Desta forma, o objetivo do agente aprendiz é aprender uma política ótima de atuação que maximize a quantidade de recompensa recebida ao longo de sua execução, independentemente do estado inicial. Assume-se que o ambiente opera segundo o modelo de processos de decisão markovianos (PDM), de modo que a decisão a ser tomada em um instante específico depende apenas das informações disponíveis nesse instante.

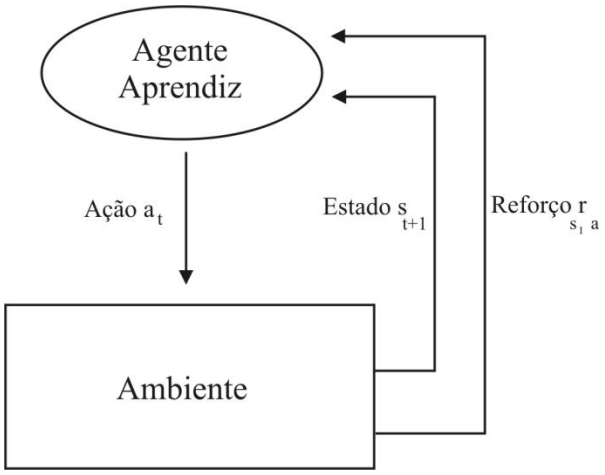


Figura 2.7. Sistema de aprendizagem por reforço.
Fonte: o autor.

A formalização do aprendizado por reforço utilizando PDM, por ser matematicamente bem estabelecido, facilita o estudo do AR.

Um PDM é definido formalmente (LITTMAN, 1994; KAELBLING; LITTMAN; MOORE, 1996; MITCHELL, 1997) pela quádrupla (*S*, *A*, *T*, *R*), onde:

S: é um conjunto finito de estados do ambiente.

A: é um conjunto finito de ações que o agente pode realizar.

$T : S \times A \rightarrow \Pi(S)$: é a função de transição de estado, em que Pi (S) é uma distribuição de probabilidades sobre o conjunto de estados S. T(st; at; st+1) define a probabilidade de se realizar a transição do estado st para o estado st+1 quando se executa a ação *at*.

$R : S \times A \rightarrow \mathbb{R}$: é a função de recompensa, que especifica a tarefa do agente, definindo a recompensa recebida por um agente ao selecionar a ação *a* estando no estado *s*.

Resolver um PDM consiste em computar a política $\Pi: S \times A$ que maximiza (ou minimiza) alguma função, geralmente a recompensa recebida (ou o custo esperado), ao longo do tempo.

2.7.2. Algoritmos genéticos

Os algoritmos genéticos (AG) apresentam uma ampla utilização em diversas áreas científicas relativas à resolução de problemas de otimização como: análise de modelos econômicos, aprendizado de máquina, problemas de engenharia, ecossistemas, sistemas imunológicos, várias aplicações na biologia como simulação de bactérias e propriedades de moléculas orgânicas [70].

Os AGs foram propostos por Holland [47] na década de 1970, que em 1975 publicou "*Adaptation in Natural and Artificial Systems*", que é considerado o ponto inicial dos AGs [47]. Holland considerou que a evolução natural é um processo consistente que produz resultados eficientes quando aplicados em problemas de otimização por meio de soluções computacionais. Os AGs geram soluções satisfatórias mesmo considerando parâmetros de inicialização aleatórios [44] e têm como peculiaridade criar descendentes através do operador de recombinação [31].

Os AGs consideram como soluções de um problema os "indivíduos" de uma "população", além disso, esses indivíduos passam por um processo de evolução a cada iteração ou "geração". Para isso, é imprescindível a construção de um modelo onde os indivíduos sejam considerados como soluções de um problema. Para ilustrar o funcionamento de um AG, na figura 2.8 são apresentados os passos necessários para a execução de um AG. Além disso, no quadro 2.2, observa-se a correspondência entre termos adotados na biologia com termos adotados em ambiente computacional.

Os AGs diferem dos métodos exatos de otimização em três aspectos: (i) trabalham a partir de uma população de soluções e não com uma única solução; (ii) não necessitam de

nenhum conhecimento derivado do problema, apenas uma forma de avaliação do resultado (função de avaliação); e (iii) utilizam regras de transições probabilísticas e não regras determinísticas.

Quadro 2.2. Comparativo entre termos computacionais e biológicos.

Termos Biológicos	Termos Computacionais
Cromossomo	Indivíduo
Gene	Caractere
Lócus	Posição do Caractere
Alelo	Valor do Caractere
Fenótipo	Interpretação do vetor de caracteres
Genótipo	Vetor de caracteres que representa o indivíduo

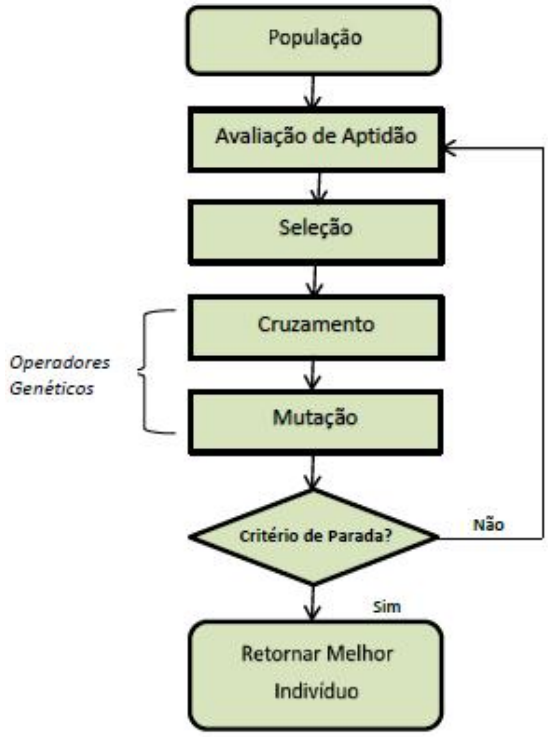


Figura 2.8. Etapas para a execução de um algoritmo genético.
Fonte: o autor.

População

Uma população de um AG é formada pelo conjunto de indivíduos cogitados como solução. Salienta-se que a definição do tamanho adequado para a população é de suma importância, pois interfere na eficiência e no desempenho dos AGs, e é o que possibilita a

diversidade genética. Uma população muito numerosa acabará exigindo maiores recursos computacionais e, com isso, prejudicará a eficiência do algoritmo devido ao aumento de tempo para analisar a função de aptidão do conjunto a cada iteração. Por outro lado, populações demasiadamente pequenas acabam perdendo a diversidade necessária para encontrar uma solução adequada, em função do espaço de soluções exploradas ser limitado, e torna-se necessário aumentar a taxa de mutação.

Indivíduos

Inicialmente, para que um AG possa ser utilizado é importante que o problema tenha uma representação de maneira que os AGs sejam capazes de tratar. Um atributo pode ser reconhecido como uma sequência de bits e o indivíduo como a concatenação das sequências de bits de todos os seus atributos [47]. Outra forma é a codificação de um indivíduo utilizando o próprio alfabeto do atributo que se precisa representar [67].

Avaliação de aptidão (*fitness*)

Conhecida como função de avaliação, esta função especifica o valor de aptidão de cada indivíduo e deve ser utilizada após gerar uma população. Ela define o quanto um indivíduo está se aproximando da solução desejada.

Seleção

O processo de seleção busca verificar o valor de aptidão de cada indivíduo em uma determinada população. Em seguida deve identificar quais possuem melhor aptidão, e os escolhidos estarão sujeitos a aplicação de operadores genéticos.

Existem inúmeras formas de seleção, porém serão destacados neste trabalho, somente os métodos de seleção por roleta e por torneio que são os mais comumente utilizados.

Na figura 2.9 é apresentado o método de seleção por roleta, em que os indivíduos com maior aptidão são representados por uma fatia maior e os indivíduos com menor aptidão são representados por uma fatia menor.

Um dos problemas da seleção por roleta é a necessidade de passagem de todos os indivíduos da população duas vezes, o que acaba ocasionando demora no processamento. No quadro 2.3, é demonstrado um exemplo deste método implementado.

INDIVÍDUO	*FITNESS*	PERCENTAGEM DO TOTAL
A		40,00
B	100	50,00
C	20	10,00
TOTAL	200	100,00

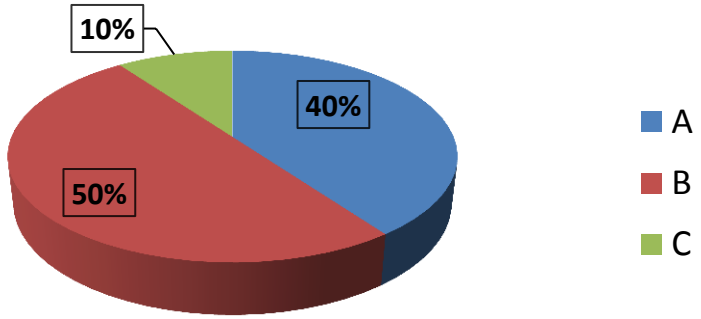


Figura 2.9. Exemplo do método de seleção por roleta.
Fonte: o autor.

Quadro 2.3. Exemplo de algoritmo do método de seleção por roleta.

```
INÍCIO
        FitSoma = somatório do valor de aptidão de todos indivíduos
        REPITA K vezes para selecionar n indivíduos
                a = valor aleatório entre 0 e FitSoma

                Buscar em todos indivíduos, acumulando em FitSomaAux o valor de aptidão de
                cada indivíduos já percorridos;

                SE FitSomaAux >= a então
                        Selecionar o indivíduo corrente;
                FIM SE
         FIM REPITA
FIM
```

O método de seleção por torneio escolhe aleatoriamente um determinado número de indivíduos de uma população formando temporariamente uma subpopulação. Depois disso, define-se uma probabilidade K – chamada de tamanho do torneio, e o indivíduo deste grupo que satisfizer esta condição será selecionado. O valor mínimo de K deve ser 2, mas para o valor máximo não há limite teórico definido [85].

Este método tem por vantagem não precisar fazer a comparação entre todos os indivíduos da população como no método da roleta. O quadro 2.4 apresenta um exemplo deste algoritmo implementado, onde n=2:

Quadro 2.4. Exemplo de algoritmo do método de seleção por torneio.

```
INÍCIO
        K = Número de indivíduos para seleção
        J = 0,75
        REPITA K vezes
                Aleatoriamente escolher 2 indivíduos da população
                a = valor  aleatório entre 0 e 1

                SE a < J
                        Selecionar o indivíduo Melhor;
                SENAO
                        Selecionar o indivíduo Pior;
                FIM SE
        FIM REPITA
FIM
```

Operadores genéticos

Os operadores genéticos são responsáveis por diversificar a população a cada geração preservando características de adaptação herdadas. Os operadores genéticos de cruzamento e mutação são essenciais neste contexto.

Cruzamento *(Crossover)*

Este operador realiza o cruzamento de determinados genes de dois indivíduos pais. O cruzamento simula a reprodução de genes em células onde partes das características de um indivíduo são recolocadas na parte correspondente do outro, gerando um novo indivíduo com a combinação das características dos pais.

Os exemplos de cruzamentos demonstrados nas figuras 2.10 e 2.11 apresentam dois tipos muito utilizados que são denominados cruzamento em um ponto e cruzamento em dois pontos. Ressalta-se que os pontos de corte podem ser fixos ou aleatórios.

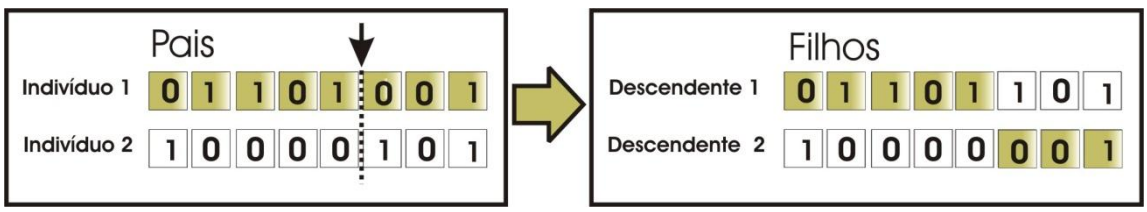


Figura 2.10. Exemplo de *crossover* em um ponto.
Fonte: o autor.

Na figura 2.10 são gerados dois descendentes selecionando um ponto de corte aleatório, onde cada filho adquire a informação genética de cada um dos pais.

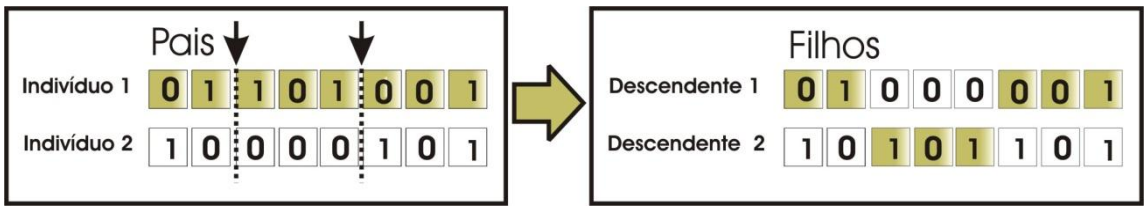


Figura 2.11. Exemplo de *crossover* em dois pontos.
Fonte: o autor.

Na figura 2.11 é realizado dois pontos de corte, onde um dos filhos receberá os genes posicionados na região central de um dos pais e o outro descendente receberá os genes que estão posicionados nos pontos extremos e vice-versa.

Mutação

A mutação é aplicada a cada indivíduo após o processo de cruzamento, alterando um ou mais genes ajudando a manter a diversidade genética da população. Essa modificação gera novos valores e permite aumentar o número de características que apareciam em baixa quantidade. Sendo assim, a mutação garante que a probabilidade de se alcançar qualquer ponto do espaço de busca não será nula. A taxa de mutação aplicada aos indivíduos geralmente é baixa – em torno de 5%. Na figura 2.12 consta um exemplo de mutação simples, alterando um gene de um indivíduo.

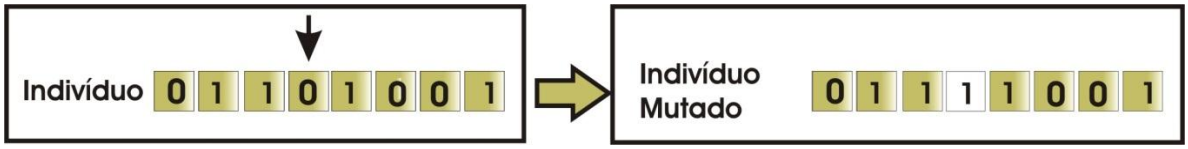


Figura 2.12. Exemplo de mutação simples.
Fonte: o autor.

2.7.3. Sistemas Classificadores

Os SCs foram introduzidos por Holland [47] e surgiram da união dos sistemas especialistas (SE) com os algoritmos genéticos. Um SE pode ser entendido como um conjunto de regras proposicionais: dada uma condição tem-se, por exemplo, uma ação. A condição forma o antecedente da regra, e a ação o seu consequente. Sistemas especialistas foram muito importantes para o avanço de inúmeras áreas, mas alguns obstáculos minaram sua ampla utilização, como o fato de todas as regras necessitarem ser fornecidas por seres humanos e manualmente. Além disso, uma vez que o conjunto de regras é estático, o sistema nunca pode descobrir se uma regra tornou-se não aplicável, consequentemente sendo necessária sua

eliminação ou modificação. Outro conflito ocorre quando mais de uma regra é aplicável a uma situação. Todos esses conflitos devem ser previstos ou o sistema pode parar e não saber como proceder [73]. Os SCs resolveram os problemas relatados dos SEs, incorporando um algoritmo genético, que no SC é o responsável pela busca e descoberta de novos conhecimentos com o consequente aperfeiçoamento do SC ao longo do tempo.

Goldberg [44] define que um SC é um sistema de aprendizagem de máquina que utiliza regras simples (chamadas classificadores) para guiar seu desempenho em um ambiente arbitrário. Um SC consiste de três componentes principais:

- Módulo de regras e mensagens;
- Módulo de atribuição de crédito;
- Algoritmo genético.

Em síntese, SC consiste de um mecanismo para criação e atualização evolutiva de regras (os classificadores) em um sistema de tomada de decisão, que codifica alternativas de ações específicas que as características de um ambiente requerem em determinado instante.

Em decorrência do(s) efeito(s) da ação verificada no ambiente, os classificadores responsáveis pelas ações são punidos ou recompensados, utilizando um fator de qualidade presente em cada classificador. Os SCs também submetem periodicamente a população de classificadores a um processo de evolução realizado utilizando os AGs, que visa descobrir novas regras, bem como remover regras com baixa qualidade.

O modo de interação do SC com o ambiente se dá com a troca de mensagens, como ilustrado na figura 2.13. As mensagens vindas do ambiente informam ao SC o estado atual do ambiente e as mensagens geradas pelo SC revelam as ações que devem ser implementadas no ambiente.

Como se observa na figura 2.13, toda comunicação do ambiente acontece por meio de "detectores" e "executores". Detectores são responsáveis por receber e codificar as mensagens recebidas do ambiente e os Executores são responsáveis pela decodificação das ações propostas pelo SC.

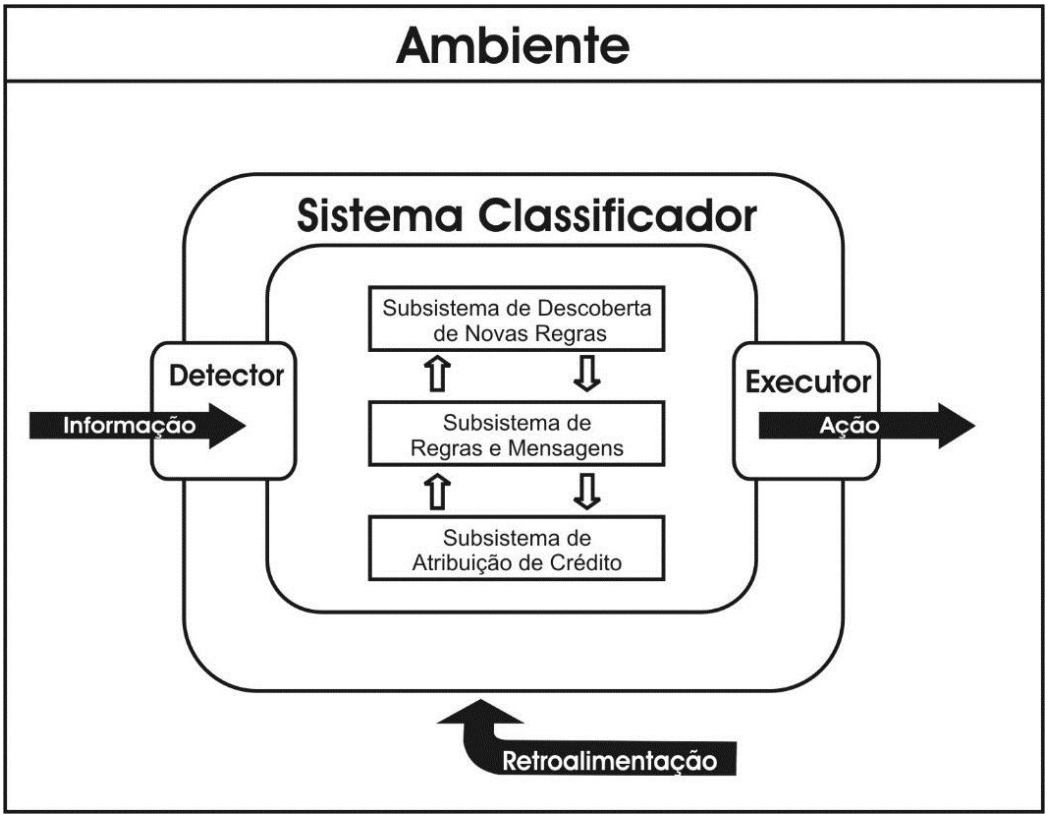


Figura 2.13. Arquitetura geral de um Sistema Classificador.
Fonte: adaptada de Goldberg [44].

Abordagens Michigan e Pittsburgh

Os SCs podem ser agrupados em duas abordagens: (i) abordagem Michigan e (ii) abordagem Pittsburgh [31], [69]. Na abordagem Michigan cada indivíduo da população representa uma única regra de classificação, ou seja, cada indivíduo (classificador) é destinado a atuar vinculado a uma única situação no ambiente. Sendo assim, a solução se dá pelo conjunto de classificadores que está sendo gerenciado e evoluído. A abordagem Michigan, foi introduzida a partir do algoritmo CS1 de Holland e Reitman [48]. Tradicionalmente, inicia-se com uma população de indivíduos criados aleatoriamente, depois é realizada uma avaliação através de aprendizado por reforço, usando algum mecanismo de atribuição de crédito de acordo com a ação executada [55].

O componente AG na abordagem Michigan tem por objetivo descobrir novas regras, bem como assegurar a coevolução da população de regras. Para isso, a energia (aptidão) do indivíduo é utilizada como critério de qualidade.

Na abordagem Michigan, como cada indivíduo codifica uma regra, esses tendem a ser sintaticamente curtos e de forma geral mais simples, reduzindo o tempo gasto para cálculo

da sua aptidão, e, por consequência, reduzindo também o tempo para recombinação e mutação [39].

No entanto, como a função de aptidão é responsável por avaliar a qualidade individual de cada integrante da população, não é trivial computar como se dá a integração entre as regras e a qualidade do conjunto, considerando toda a população.

Também existe o problema relacionado aos AGs, que normalmente convergem para um único indivíduo, o que é indesejável, devido a cada indivíduo tratar de um ponto específico e o que se espera são regras que possam lidar com várias situações e não somente uma. Como forma de tratar esses potenciais problemas, novos esquemas de substituição da população e esquemas de nicho estão sendo propostos e utilizados para possibilitar maior diversidade de regras na população [37], [44].

Uma contribuição importante para os SCs veio do algoritmo XCS [54], que define um conceito que valoriza regras que maximizam a acurácia e a generalização. SCs baseados em Holland [47] tem o valor de aptidão usado no processo de atribuição de crédito igual ao valor de aptidão utilizado no AG. De outra forma, o XCS utiliza um mecanismo mais sofisticado, que considera como valor de aptidão do AG um valor baseado na precisão da predição.

Em outra abordagem chamada Pittsburgh cada indivíduo é um conjunto de regras que codifica uma solução candidata completa para o problema. O cálculo do valor de aptidão de cada indivíduo é medido pelo desempenho de todo o conjunto de regras. De forma geral, para atribuição do valor de aptidão de cada indivíduo apenas a precisão de classificação é considerada.

Na abordagem Pittsburgh a cooperação entre indivíduos não é necessária, posto que um indivíduo já é um conjunto completo de regras e, consequentemente, técnicas de nicho também não são necessárias. Em Jong e Spears [51] é proposto o algoritmo GABL onde os indivíduos representam regras de tamanho fixo, codificados como uma *string* binária e o valor de aptidão de cada indivíduo se baseia apenas na precisão de classificação. Um ponto crítico, observado em trabalhos que utilizam essa abordagem, é que os indivíduos são mais longos sintaticamente, o que demanda maior poder computacional para o processo de avaliação de aptidão de cada indivíduo [37].

Nesta tese, devido à natureza do problema e aplicação que o SC será utilizado, optou-se em utilizar a abordagem Michigan.

Regras e Energia

Assim como as regras de um Sistema Especialista (SE), os classificadores são compostos por uma parte antecedente e outra consequente (podendo um classificador possuir duas ou mais condições na parte antecedente), como ilustrado no quadro 2.5.

Na proposta original de Holland [47], que utilizava codificação binária, a parte antecedente do SC é um vetor de tamanho fixo, formado pela concatenação dos caracteres “0”, “1” e “#”, os quais são elementos do conjunto que compõe o alfabeto ternário {0,1,#}. O caractere “#’ conhecido como o símbolo *“don’t care”* pode valer tanto “1” como “0”, dependendo da situação, permitindo a existência de regras genéricas, sendo que, quanto mais símbolos "#" estiverem presentes em uma regra mais genérica ela será. Portanto, se um classificador possui todos os caracteres de seu antecedente iguais a "#", sua especificidade é zero; caso não haja nenhum destes símbolos, a especificidade será máxima.

Como parâmetro de qualidade, todo classificador possui um campo que representa sua energia específica, que é determinada de acordo com o desempenho médio e utilidade dentro da população de classificadores ao longo de sua atuação, considerando a realimentação recebida para cada ação aplicada no ambiente. Portanto, quanto maior a energia (aptidão) de um classificador melhor terá sido seu desempenho no ambiente, e maior será sua probabilidade de atuar novamente no ambiente e de perpetuar suas características em outros classificadores nas futuras gerações, caso os objetivos e as características do ambiente permaneçam as mesmas.

Quadro 2.5. Exemplo de classificadores binários.

Código Classificador	**Classificadores ou Regras (SE): (ENTAO)**	**Energia**
1	1#1##: 11	8,5
2	1110#: 01	15,2
3	11111: 11	5,9
4	##0##: 10	19,0

Sistema de tratamento de regras e mensagens

O subsistema de tratamento de regras e mensagens é utilizado quando os detectores capturam alguma mensagem do ambiente. Após os detectores enviarem a mensagem realiza-se a codificação da mensagem de forma que o SC possa reconhecê-la e inseri-la em um processo de "comparação".

Na fase de comparação, todos os classificadores tentam identificar sua parte antecedente com a mensagem. A identificação pode ser feita por comparação bit a bit, ou pelo cálculo demonstrado na equação 2.2 - variante da distância de *Hamming* [72], onde *l* representa o comprimento do vetor de bits e *n* corresponde à quantidade de 0's e 1's que não se combinaram. Segundo Booker [72], este cálculo permite que indivíduos mais específicos em cromossomos muito longos tenham chance de competir com indivíduos menos específicos.

$$M = \frac{l - n}{l^2} \qquad \text{(Exp. 2.2)}$$

Outra forma de comparar os classificadores é comparando cada bit/campo ao antecedente da regra. Aqueles que forem equivalentes são ativados e passam a concorrer ao direito de atuar no ambiente, conforme ilustrado na figura 2.14 e, em seguida, serão enviados ao subsistema de apropriação de crédito.

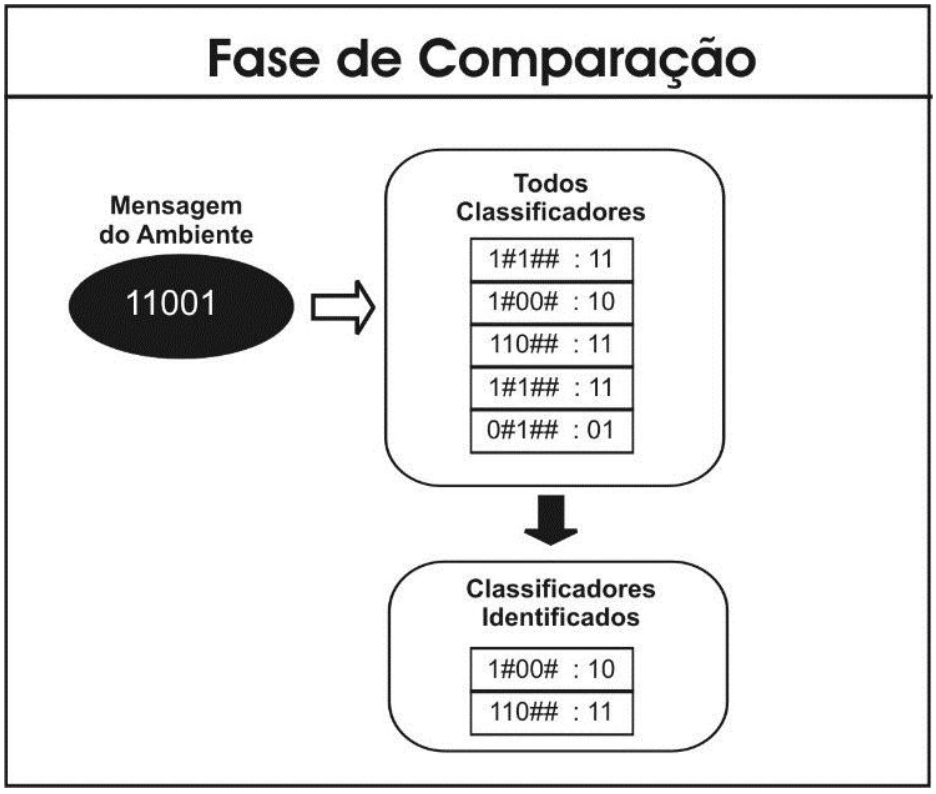


Figura 2.14. Processo de comparação de classificadores com uma mensagem do ambiente.
Fonte: o autor.

Alguns SCs permitem classificadores postadores de mensagens, cuja parte consequente da regra (ação) é basicamente postar uma nova mensagem no ambiente. Desta forma, além das mensagens decorrentes dos eventos do ambiente, haverá mensagens dos próprios classificadores.

Subsistema de apropriação de crédito

O módulo de atribuição de crédito é de fundamental importância em um sistema de aprendizagem, onde várias partes atuam para determinar o desempenho global deste sistema. Em ambientes complexos, onde existe a possibilidade de vários classificadores serem ativados em paralelo, é importante identificar se as ações de todos os classificadores foram adequadas [63].

Entre os diversos algoritmos de atribuição de crédito existentes, o algoritmo mais disseminado e conhecido é o *Bucket Brigade*. Este algoritmo é utilizado em SCs, que contemplam classificadores postadores de mensagens, ou seja, classificadores que ativam outros classificadores, sendo capazes de recompensar ou punir todos os que levaram a uma determinada modificação do ambiente, mesmo que isso tenha sido detectado após uma sequência de atuações de classificadores.

Para SCs que não utilizam classificadores postadores de mensagem, como é o caso do SC desenvolvido nesta tese, uma opção é o algoritmo que segue a abordagem de estímulo-resposta (E-R), onde somente um classificador é ativado para atuar no ambiente, simplificando o processo de recompensa ou punição [73].

Cálculo da aposta

Ao ser enviada uma mensagem de entrada ao SC, verifica-se se na população de classificadores acontece a combinação (ou *matching*) de algum classificador com a mensagem de entrada, ou seja, se ocorre *matching* de cada caractere da mensagem de entrada, com o caractere correspondente do antecedente do classificador, sendo que o caractere # do antecedente do classificador sempre garante *matching* para determinado gene. O cálculo de quanto específico é um classificador é dado na expressão 2.3, onde é calculada a especificidade (S) de um classificador.

$$S = (na - nt)/na \qquad \text{(Exp. 2.3)}$$

onde:

na é o comprimento do antecedente;

nt é o número de # do antecedente.

Caso existam mais de um classificador que obteve *matching* com a entrada, estes participam de uma competição efetuando uma aposta dada pela expressão 2.4. O classificador com maior aposta é o ganhador da competição.

$$B = k0*(k1+ k2*S*k3)*E \qquad \text{(Exp. 2.4)}$$

onde:

B: Aposta do classificador;

k0: Coeficiente de aposta, valor positivo menor ou igual a 1;

k1: Valor positivo, menor ou igual a 1, que corresponde à participação da parte não referente à especificidade na aposta;

k2: Valor positivo, menor ou igual a 1, correspondente à participação da especificidade na aposta;

k3: Parâmetro controlando a importância da especificidade para determinar a aposta (padrão = 1);

S: Especificidade;

E: Energia(aptidão) do classificador.

Taxa de aposta

O classificador vitorioso é aquele que oferece maior aposta, e este valor é integralmente subtraído de sua força. Os outros classificadores que participaram da competição multiplicam a sua aposta pela taxa de aposta, **TaxaBid**, antes de fazerem essa subtração. Assim, a energia dos classificadores após a aposta é calculada pelas expressões:

$E(t+1) = E(t) - B$ Para o classificador ganhador (Exp. 2.5)

$E(t+1) = E(t) - TaxaBid*B$ Para os outros apostadores (Exp. 2.6)

onde:

TaxaBid: Taxa de aposta

t iteração em processamento

Taxa de vida

Finalmente, em cada iteração todos os classificadores ficam sujeitos a um decréscimo em sua energia em virtude da taxa de vida determinada por:

$$TxVida = 1 - (1/2)^{(1/n)} \qquad \text{(Exp. 2.7)}$$

onde:

n: vida média, medida em número de iterações.

Recompensa ou punição

Nas etapas anteriores, em razão da competição, os classificadores que participaram tiveram que pagar uma taxa de aposta, e também ficaram sujeitos a uma taxa de vida. Neste ponto, ainda falta colher a retroalimentação (*feedback)* do ambiente que verificará se a ação resultante do consequente do classificador ganhador teve um efeito positivo ou negativo. Caso o resultado tenha sido positivo, o mesmo será recompensado, caso contrário sofrerá penalização. Quando o classificador é recompensado será restituída sua aposta, além de ser somada a recompensa a sua energia.

Segue as expressões referentes à obtenção da energia pela aplicação do mecanismo de recompensa e punição:

$$E = E + B + R \rightarrow \text{Para recompensa} \qquad \text{(Exp. 2.8)}$$

$$E = E - P \rightarrow \text{Para punição} \qquad \text{(Exp. 2.9)}$$

Subsistema de descoberta de novas regras

Ao final de cada época de iterações, espera-se que a energia dos classificadores tenha sido devidamente ajustada pelo subsistema de atribuição de crédito, para então iniciar a evolução das regras, chamando o subsistema de descoberta de novas regras.

Esta fase utiliza técnicas de computação evolucionária visando buscar uma população de classificadores cada vez mais adaptada ao ambiente, de acordo com os objetivos a serem atendidos.

Em resumo, um algoritmo genético é utilizado para produzir uma nova geração de classificadores, pela aplicação dos operadores de seleção, *crossover* e mutação, em uma parte da população original. A probabilidade de seleção de um classificador é proporcional a sua energia, no entanto, todos os classificadores podem ter a chance de ser selecionados.

Após atingir o número de gerações definido no AG, os filhos gerados na última geração serão inseridos na população em substituição a outros indivíduos, conforme ilustrado na figura 2.15.

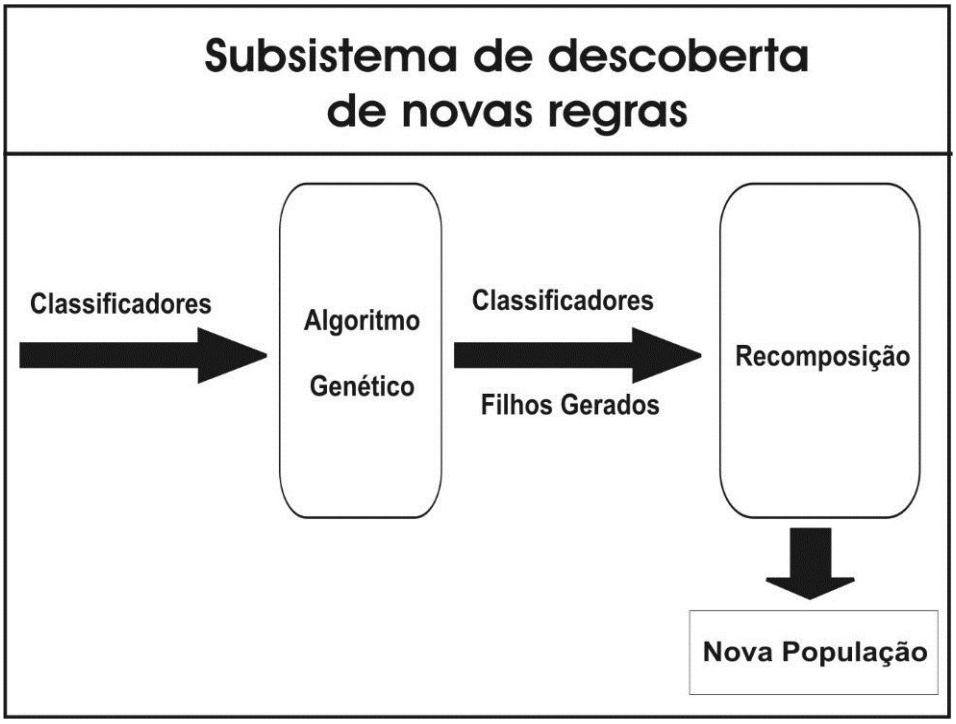


Figura 2.15. Processo evolutivo e recomposição da população de um SC.
Fonte: o autor.

Capítulo 3

Trabalhos Relacionados

Este capítulo visa fazer uma revisão de trabalhos relacionados ao *tuning* de projeto físico de bancos de dados, sobretudo no que tange ao *tuning* de índices, destacando suas características, contribuições e deficiências.

3.1. Considerações iniciais

Visando obter um bom desempenho, uma preocupação constante de diversos fabricantes de SGBDRs diz respeito ao *autotuning* de SGBDs.

Em meados de 1997, a empresa Microsoft iniciou o projeto *AutoAdmin* que culminou com o desenvolvimento da ferramenta *Index Tuning Wizard* para o SGBD SQL Server 7. Baseados especialmente nessa experiência, Chaudhuri e Narasayya [78] discutiram os algoritmos utilizados pelo projeto *AutoAdmin* e os avanços na área de *autotuning* obtidos no período de 1997 a 2007. A IBM, também interessada em prover soluções para *autotuning* de SGBDs, iniciou em 2002 o projeto SMART [25], visando enriquecer o gerenciador de seu SGBD DB2 l Universal Database, com características automáticas. A Oracle, por sua vez, iniciou atividades na área de *autotuning* em seu SGBD, desde a versão 10g, oferecendo

recursos para ajuste automático por meio da ferramenta ADDM (*Automatic Database Diagnostic Monitor*) [29], [32].

Em razão de existirem várias abordagens para o *tuning* de projeto físico, é possível identificar algumas formas de classificá-las [9], como: (i) contínuo ou não contínuo; (ii) autônomo ou não autônomo; (iii) intrusivo ou não intrusivo. Abordagens caracterizadas como contínuas ou não contínuas, dependem de possuírem ou não a capacidade de capturar e analisar dinamicamente uma carga de trabalho. Já as abordagens autônomas têm a capacidade de atuar realizando o *tuning* de projeto físico de forma automática, por meio da criação, exclusão ou reconstrução de estruturas do banco de dados. Por outro lado, as ferramentas não autônomas transferem para o DBA essa responsabilidade. Abordagens classificadas como intrusivas, por sua vez, são aquelas que exigem mudanças que são fortemente acopladas ao código fonte de um SGDB utilizado, restringindo a sua utilização a outros SGBDs.

O quadro 3.1 apresenta um resumo comparativo entre os estudos que serão discutidos neste capítulo e que motivaram o desenvolvimento desta tese.

As características presentes no quadro 3.1, que será utilizado para diferenciar as abordagens discutidas, são as seguintes:

- tipo de estrutura de acesso: indica os tipos de estruturas de acesso utilizadas pela abordagem (índice primário, índice secundário, visões materializadas, partições de tabelas, etc.) e a organização física dessas estruturas;
- ações de gerenciamento: representam as ações de gerenciamento executadas sobre as estruturas de acesso utilizadas, ou seja, as ações que são suportadas pela solução estudada, que podem ser: criação, remoção e reorganização de estruturas de acesso;
- SGBDs suportados: lista os SGBDs suportados pela referida solução;
- modelo de custos: indica se o modelo de custos utilizado pelas heurísticas é interno (o próprio modelo de custos utilizado pelo SGBD) ou externo (modelo de custos independente do SGBD);
- avaliação de desempenho: descreve se a solução adotou algum *benchmark* e, em caso positivo, quais *benchmarks* foram utilizados para a avaliação de desempenho da solução;
- considera discos SSDs: indica se o trabalho considera as novas características dos discos SSDs;
- técnica *on-line*, *off-line* ou semiautomática.

Quadro 3.1 Apresentação sumarizada dos trabalhos relacionados.

Trabalho Analisado	**Tipo de estrutura de acesso**	**Ações de gerenciamento**	**SGBDs suportados**	**Avaliação de desempenho**	**Considera discos SSDs**	**Técnica *online*, semiautomática ou *off-line***
Lohman et al. [10]	Índice Sec., árvore B+	Criação	DB2	TPC-C	Não	*Off-line*
Agrawal et al.[36]	Índice Sec., árvore B+	Criação	Sql Server	TPC-H	Não	*Off-line*
Schnaitter et al. [62]	Índice Sec., árvore B+	Criação e Remoção	PostgreSQL	*	Não	*On-line*
Luhring et al. [4]	Índice Sec., árvore B+	Criação e Remoção	DB2	TPC-H	Não	*On-line*
Bruno e Chaudhuri [35]	Índice Sec., árvore B+	Criação e Remoção	Sql Server	TPC-H	Não	*On-line*
Monteiro [11]	Índice Prim./ Sec., árvore B+	Criação, Remoção e Reorganização	Qualquer SGBDR	TPC-H	Não	*On-line*
Maier et al. [26]	Índice Sec., árvore B+, partições de tabelas	Criação, Remoção e Reorganização	Qualquer SGBDR	SDSS DR4 dataset	Não	*On-line*
Jimenez et al. [8]	Índice Sec., árvore B+	Criação e Remoção	PostgreSQL	*	Não	Semiautomática
EDB, Index Advisor[16]	Índice Sec., árvore B+	Criação	PostgreSQL	*	Não	*Off-line*
PostgreSQL Workload Analyser [17]	Índice Sec., árvore B+, Brin	Criação	PostgreSQL	*	Não	*Off-line*
Almeida et al. [9]	Índice Prim./ Sec., árvore B+	Criação, Remoção e Reorganização	Qualquer SGBDR	TPC-H	Não	*On-line*
Sharma et al. [53]	Índice Sec., árvore B+	Criação	PostgreSQL	TPC-H	Não	*Off-line*

** Testes próprios*

3.2. Abordagens *off-line*

3.2.1. *Database Tuning Advisor for Microsoft SQL Server 2005*

No contexto do projeto *AutoAdmin*, vários trabalhos se destacaram culminando com o desenvolvimento da ferramenta *Index Tuning Wizard* para o SGBD SQL Server 2005, que, a partir de uma análise de uma carga de trabalho obtida pelo DBA, sugere a criação de um conjunto de índices. Em Agrawal et al. [36] são caracterizados os avanços ocorridos para a

evolução da ferramenta. Na qual, além da seleção automática de índices, é realizada também a seleção de visões materializadas e o particionamento de grandes tabelas.

3.2.2. *DB2 Advisor: An Optimizer Smart Enough to Recommend its Own Indexes*

Integrado ao projeto SMART, da IBM, Lohman et al. [10] apresentam uma ferramenta de seleção de índices para o SGBD DB2 baseada no clássico problema da mochila. Neste trabalho, utiliza-se o próprio otimizador do SGBD para enumerar os melhores índices para uma carga de trabalho e disponibiliza recursos de restrições de disco e uma heurística para a seleção de índices. Nesse sentido, o otimizador é estendido com um modo de sugestão de índices e, antes da otimização de uma determinada consulta SQL, índices hipotéticos são gerados para todas as colunas relevantes. Em seguida, os índices recomendados para a cláusula SQL são utilizados como entrada em uma heurística de seleção de índices que tenta encontrar o melhor conjunto de índices para a carga de trabalho como um todo.

3.2.3. *Enterprise DB Index Advisor*

Desenvolvida pela empresa Entreprise DB [16], essa ferramenta comercial de auxílio ao DBA, ao receber uma consulta SQL, faz sugestões de quais colunas devem ser indexadas, mas somente para índices implementados em árvores B+ (coluna única ou multicoluna). Internamente são utilizados índices hipotéticos para estimar os custos da presença ou não de um índice, como se tais índices estivessem disponíveis no SGBD. As duas principais formas de usar a ferramenta são: (i) invocando um programa utilitário, fornecendo um arquivo de texto com as consultas a partir das quais o *Index Advisor* irá gerar um arquivo com as sugestões, no formato de comandos CREATE INDEX e; (ii) executando as consultas, via programa utilitário EDB-PSQL, sendo as sugestões armazenadas em uma tabela específica do PostgreSQL, sendo possível acessá-la via comandos SQL [16].

3.2.4. *PostgreSQL Workload Analyser* (POWA)

POWA [17] é um projeto aberto sob a licença PostgreSQL. A ferramenta pode ser instalada como uma extensão do SGBD PostgreSQL (similar ao AWR do SGBD Oracle), que, por sua vez, utiliza outra extensão chamada *pg_qualstats*. Esta última é responsável por descobrir quais os predicados mais executados, em uma carga de trabalho SQL e, desta forma, a ferramenta é capaz de sugerir o menor conjunto de índices para otimizá-la.

POWA recorre a heurísticas internas com ênfase à sugestão de índices multicolunas e as sugestões de índices podem ser visualizadas por meio de uma interface gráfica. É possível visualizar também o tempo de execução de consultas, blocos mais acessados, blocos temporários e em cache, bem como o consumo de CPU durante períodos especificados pelo usuário. Um procedimento obrigatório para utilização da ferramenta POWA é executar uma mesma carga de trabalho, duas ou mais vezes, para se obter as recomendações de índices [17].

3.2.5. *The Case for Automatic Database Administration using Deep Reinforcement Learning*

Sharma et al. [53] propõem uma abordagem para *tuning* de índice chamada NoDBA, que utiliza a abordagem, aprendizado profundo por reforço, também conhecida como: *Deep Reinforcement Learning*.

Destaca-se que, em contraste com o aprendizado supervisionado tradicional [80], onde, por exemplo, uma rede neural é treinada utilizando um conjunto de dados e um conjunto de resultados esperados, na aprendizagem por reforço o processo de treinamento não requer resultados esperados. Em vez disso, o processo é completamente dirigido pelas chamadas recompensas, indicando se ações tomadas para uma entrada levaram a um resultado positivo ou negativo. Dependendo do resultado, a rede neural é encorajada ou desencorajada a considerar a ação sobre essa entrada no futuro. Os autores salientam que a abordagem utilizando aprendizagem por reforço profundo, de forma geral, é similar ao comportamento de um DBA que, normalmente, tomará futuras decisões com base nas experiências já realizadas.

A utilização da rede neural no *tuning* de SGBDs requer a realização do treinamento da rede neural. Para isso, os autores definiram, inicialmente, um ambiente chamado problema, que consiste nos quatro componentes a seguir.

(1) Entrada para a rede neural. Essa é tipicamente a carga de trabalho atual na forma de características de consulta, para a qual o sistema deve ser otimizado, bem como o estado atual da configuração.
(2) Conjunto de ações que podem ser tomadas. Uma ação poderia ser criar um índice secundário em uma determinada coluna de uma tabela ou alterar o tamanho de um *buffer* de banco de dados. Tal ação é uma transição da configuração atual do sistema para uma nova configuração do sistema.

(3) Função de recompensa. Para classificar o impacto de uma ação tomada, é calculado, por exemplo, o tempo ou o custo de execução da nova configuração após a ação ter sido tomada e, posteriormente, comparado com a melhor configuração obtida. Quanto maior a melhoria, maior é a recompensa positiva retornada. Se o desempenho diminuir, uma recompensa negativa será retornada.

(4) Hiper-parâmetros para direcionar a aprendizagem. Isso inclui propriedades da rede neural (por exemplo, número de camadas ocultas, número de nós por camada), bem como propriedades do processo de aprendizado, como o número de iterações.

Embora o trabalho de Sharma et al. [53], tenha sido aplicado somente para *tuning* de índices, os autores enfatizam que a abordagem pode ser aplicada a outros problemas de *tuning*, como, por exemplo, ao processo de otimização de consultas.

3.3. Abordagens *on-line*

3.3.1. *On-Line Index Selection for Shifting Workloads*

Em Schnaitter et al. [62] é proposto o *framework* COLT, o qual é capaz de monitorar continuamente a carga de trabalho de um SGBD e sugerir índices eficazes, de acordo com a carga de trabalho submetida. COLT coleta estatísticas em diferentes níveis de detalhes sobre índices reais e índices hipotéticos.

Além disso, esse *framework* usa heurísticas para automaticamente regular o seu próprio desempenho, diminuindo sua atuação quando o SGBD está bem ajustado ou aumentando-a quando ocorrem mudanças na carga de trabalho que implicam na necessidade de reajustar a configuração de índices do SGBD.

Na Figura 3.1, são ilustrados os componentes principais da ferramenta COLT, onde é possível verificar também que foi necessário inserir novos módulos no código fonte do SGBD PostgreSQL. O módulo EQO (*extended query optimizer*), principal componente do *framework*, tem o objetivo de substituir o otimizador de consultas do PostgreSQL. A responsabilidade precípua do EQO continua sendo a mesma do otimizador padrão do PostgreSQL: a seleção de um plano de execução ótimo para uma determinada consulta recebida como entrada. No entanto, o EQO tem habilidades extras, como a de trabalhar com índices materializados e hipotéticos. Mais especificamente, dado uma consulta *q* e um índice

i, o EQO retorna o ganho no custo da consulta *q* caso o índice *i* seja utilizado, denotado pela expressão 3.1.

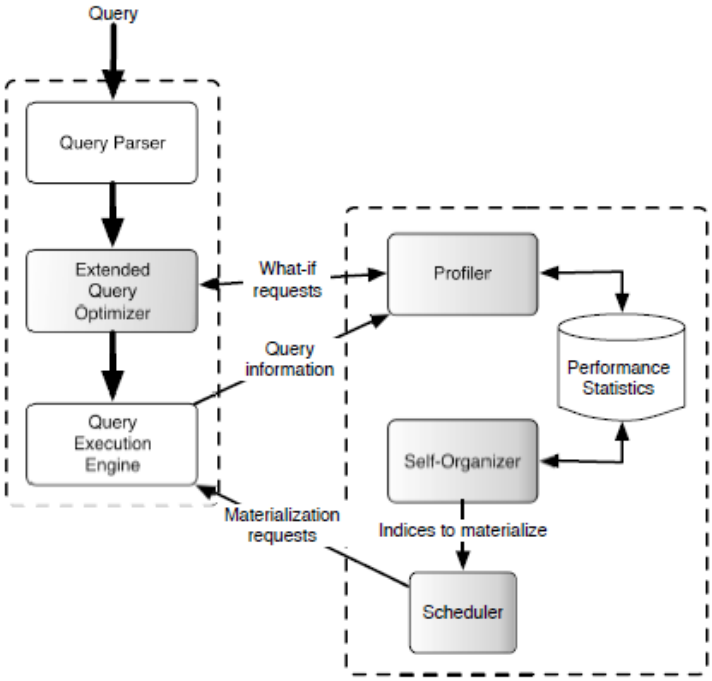


Figura 3.1. Arquitetura da ferramenta COLT.
Fonte: Schnaitter et al.[62].

$$gain(i, q) = |CM - Ci| \qquad \text{(Exp. 3.1)}$$

Sendo assim, considere que CM seja o custo do plano de execução inicial. Durante o processo de otimização de cada consulta recebida, o EQO calcula o ganho para cada índice existente, seja ele materializado ou hipotético. Em seguida o EQO recebe um conjunto de índices P. Para cada índice *i*, o EQO computa um novo plano de execução, denotado por C*i*, da seguinte maneira:

- se *i* é um índice materializado, o EQO gera o melhor plano de execução que não utiliza *i*;
- se *i* é um índice hipotético, o EQO gera o melhor plano de execução considerando a possibilidade de utilizar o índice *i*.

A esses planos de execução os autores denominam de "*what-if plans*". Dessa forma, o ganho do índice *i* para a consulta *q*, é determinado pela diferença de custo entre CM e C*i*, conforme exp. 3.1

Outros módulos do *framework* que podem ser observados na figura 3.1 são:

- *Profiler*: responsável por obter estatísticas de desempenho de índices candidatos que são atualizados, incrementalmente, após a avaliação de cada consulta;
- *Self Organizer* (SO): componente ativado no final de cada época. SO verifica as informações estatísticas obtidas pelo módulo *Profiler* e prevê o benefício esperado de cada índice, considerando a carga de trabalho;
- *Scheduler*: módulo responsável por escolher o melhor momento para se materializar um índice. Várias estratégias de escalonamento são possíveis como: (1) materialização imediata; (2) materialização de índices durante o tempo ocioso do sistema; e (3) utilização de resultados intermediários de futuras consultas para construção de índices mais eficientes.

Outra característica importante é a capacidade do *framework* para gerenciar restrições do espaço disponível para a materialização de índices.

Um conceito introduzido na implementação do COLT é a divisão das consultas recebidas em intervalos regulares denominados "época", que equivale a 10 (dez) consultas. Durante uma "época" são coletadas estatísticas sobre a carga de trabalho e os ganhos dos índices candidatos. Ao final da "época" analisa-se a necessidade de alteração na configuração de índices (materializados e hipotéticos), podendo ocorrer a promoção dos índices hipotéticos em materializados, e os índices materializados permanecerem ou serem removidos (voltando à situação de hipotéticos).

Concernente ao conjunto de índices materializados e hipotéticos é realizada uma subdivisão em três conjuntos:

- conjunto M: representa o conjunto dos índices materializados que são gerenciados pelo STM;
- conjunto H: *(hot set*) contém os índices candidatos que foram relevantes para as consultas recentemente analisadas e que mostraram fortes evidências de sua grande utilidade para elevar o desempenho da carga de trabalho;
- conjunto C: (*cold set*) contém os índices candidatos que foram relevantes para as consultas recentemente analisadas, mas cuja utilidade para a carga de trabalho se mostrou apenas razoável.

Para realizar a distribuição dos índices candidatos entre os conjuntos M, H e C é necessário calcular o ganho individual de cada índice. Essa atividade mostra-se computacionalmente cara, pois para calcular o ganho de determinado índice para uma

consulta é preciso otimizá-la mais uma vez. Dessa forma, adicionou-se um limite para o número de otimizações efetuadas em uma "época". No entanto, esse limite pode variar, dependendo da estabilidade da carga de trabalho em que o sistema está trabalhando no momento [62].

3.3.2. *Autonomous Management of Soft Index*

Luhring et al. [4] utilizam uma abordagem para a manutenção e criação automática, que também utiliza o recurso de índices hipotéticos (denominados *Soft Indexes*), monitorando e coletando estatísticas acerca dos índices reais e hipotéticos.

É utilizado também o modelo genérico de sintonização automática proposto inicialmente em Weikum et al. [56], o qual consiste no ciclo de observação, predição e reação. Essas fases são repetidas, continuamente, com o intuito de tomar decisões (fase de reação) a partir do monitoramento e análise do comportamento do sistema.

Cada consulta Q submetida ao SGBD é otimizada duas vezes, uma sem considerar índice algum e outra levando em conta todos os índices candidatos. O benefício do conjunto de índices candidatos é calculado atribuindo a cada índice candidato parte do benefício total, empregando aproximação, conforme estratégias apresentadas em Sattler et al. [64].

Para operações de atualização (*update*) que são considerados um malefício para o sistema [4], são calculadas estimativas do número de linhas afetadas por esses comandos, a altura da árvore e um fator derivado empiricamente que represente o custo de atualização de uma entrada nos índices.

Sobre o aspecto temporal da solução, é utilizado o conceito de "época" (SATTLER et al.[64], que delimita a duração de um período de observação. O limite pode ser em termos do número de consultas, por um benefício global máximo de algum índice candidato ou por número máximo de recomendações para um índice.

Na etapa de reação é realizada a criação contínua e *on-line* de índices. Na fase anterior, se detectada a necessidade de alterar a configuração de índices, são criados *deferred indexes*, ou seja, índices prontos para posterior materialização.

No trabalho, são investigados os ganhos de desempenho obtidos com a integração da criação de índices com o processamento de consultas, o que possibilita, por exemplo, a construção de um índice durante a execução de um *Table Scan*. Nesse sentido, foram introduzidos dois novos operadores: *IndexBuildScan* e *SwitchPlan*. O operador

IndexBuildScan estende a operação *Table Scan* sobre uma tabela para criar um ou mais *deferred indexes*. O operador *IndexBuildScan* recebe como parâmetro a lista de *deferred indexes* definidos sobre uma tabela, os quais serão construídos durante a varredura na tabela.

O operador *SwitchPlan* tem por objetivo permitir o uso de um índice (recém-criado) na mesma consulta em que foi projetado/criado. A figura 3.2 demonstra como se comporta uma operação, utilizando o operador *SwitchPlan*. Aponta que, durante a primeira fase, as tuplas são afetadas pelo operador esquerdo e, no entanto, durante as fases seguintes o operador *SwitchPlan* alterna, para, em vez de fazer uma varredura completa (*tablescan*) executar uma varredura envolvendo a criação e utilização de um índice.

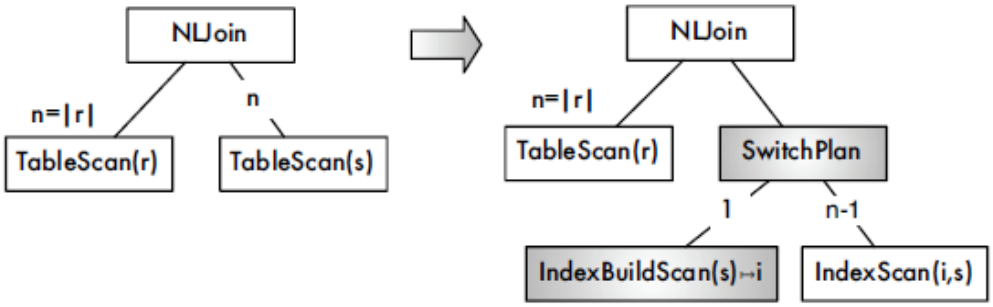


Figura 3.2. Exemplo utilizando os operadores SwitchPlan e IndexBuildScan.
Fonte: Luhring et al.[4].

A solução proposta foi implementada no SGBD PostgreSQL 7, o qual necessitou ser estendido para a criação dos novos operadores: *SwitchPlan* e *IndexBuildScan*.

3.3.3. *An On-line Approach to Physical Design Tuning*

Em Bruno e Chaudhuri [35] é apresentada uma ferramenta de sintonização automática de índices que foi implementada como um componente do Microsoft SQL Server. É executada continuamente e reage modificando de forma automática o projeto físico do banco de dados de acordo com variações na carga de trabalho e dados.

As principais características da abordagem são:

- ao receber uma consulta, em tempo de execução, identifica-se um conjunto de índices candidatos relevantes e calcula-se o quanto de desempenho foi perdido pela ausência daquele possível índice;

- após reunir informações suficientes sobre os benefícios de um possível índice, o mesmo é automaticamente criado ou também pode haver caso de remoções, quando identificados malefícios da presença de algum índice.

Para prover o recurso de índices hipotéticos, o gerenciador de metadados foi estendido para inclusão de um conjunto de contadores, responsáveis por computar o benefício de cada índice para a carga de trabalho. Essa medida é utilizada para selecionar índices com potencial de ser materializados ou excluídos do SGBD.

Salienta-se ainda que, havendo uma restrição de armazenamento secundário que impeça todos os índices selecionados de serem materializados, a ferramenta decide quais deles materializar, levando em conta se a eliminação de um índice pode liberar espaço para dois ou mais cuja soma dos benefícios seja maior que a do índice eliminado, bem como se é possível juntar dois ou mais índices.

Como característica importante, a solução permite que após a execução de uma ou várias consultas reduza-se a carga do processo de sintonização automática. Apesar de ser uma solução que não exige intervenção do DBA, este pode monitorar o estado interno e intervir conforme preferir.

A solução e algoritmos propostos apresentam baixa sobrecarga, além de considerar o custo da atualização de índices, causadas por comandos de *update*, custo de criação de tabelas auxiliares e temporárias, bem como as restrições no espaço de armazenamento secundário.

3.3.4. Uma abordagem não intrusiva para a manutenção automática do projeto físico de bancos de dados

No trabalho de Monteiro [11], tem-se uma abordagem não intrusiva para a manutenção automática do projeto físico de bancos de dados que utiliza agentes de *software*. O autor propõe um mecanismo completamente desacoplado do código do SGBD utilizado (não intrusiva), podendo ser utilizado com qualquer SGBD e também *on-line*, com execução independentemente de interações com os DBAs.

A solução traz uma heurística integrada para a seleção e acompanhamento de índices – denominada heurística HISAI, a qual se baseia no conceito de otimização hipotética. Essa heurística realiza, em um único passo, a seleção de índices candidatos (hipotéticos), o acompanhamento dos índices candidatos e reais, além do acompanhamento do nível de fragmentação dos índices reais.

Outro recurso presente é a manutenção de índices primários e secundários e o acompanhamento do nível de fragmentação das suas estruturas. Logo que detectado um índice que se encontra num grau de fragmentação superior ao limite configurado (pelo DBA) esse índice é automaticamente reorganizado. Assim, evita-se a ocorrência de problemas de desempenho advindos da existência de índices fragmentados.

3.3.5. *PARINDA: An Interactive Physical Designer for PostgreSQL*

Em Maier et al. [26], é proposta a ferramenta PARINDA, a qual é capaz de automatizar o projeto físico de SGBDRs por meio da criação de índices e partições de tabelas.

Diferentemente de outras abordagens [13], [40] e [49], que utilizam heurísticas gulosas para eliminar o espaço de busca, a abordagem proposta não afasta do espaço de busca possíveis soluções candidatas, mas pesquisa todos possíveis candidatos úteis antes de sugerir o conjunto ideal de recursos.

Para isso, a ferramenta primeiro modifica o otimizador para habilitar a característica denominada "*what-if*", onde recursos de projeto físicos são simulados por meio da criação de estatísticas no catálogo do SGBD.

Dessa forma, como o otimizador de consulta atua principalmente com estatísticas, ele não pode diferenciar os recursos de projeto reais dos que são hipotéticos (simulados). Portanto, essas estruturas *what-if* permitem que o DBA estime o benefício que obteria se as estruturas estivessem realmente presentes no banco de dados.

Para realizar a busca pelo melhor conjunto de partições de forma automática, emprega-se a técnica usada na ferramenta AutoPart [77]. Além disso, é realizada a reescrita das consultas de entrada para corresponder às partições sugeridas.

Para encontrar o conjunto ideal de índices, os autores utilizam a técnica ILP [46] que constrói um programa linear inteiro e resolve-o usando um solucionador combinatório padronizado.

3.3.6. *Automatic Physical Design Tuning based on Hypothetical Plans*

No trabalho de Almeida et al. [9] é apresentada uma abordagem contínua, autônoma e não intrusiva para a manutenção do design físico, denominada *HypoPlans*. Assim como em outras abordagens, o trabalho utiliza configurações hipotéticas para realizar o *self-tuning* de

projeto físico, sendo apresentado no trabalho experimentos envolvendo o problema de seleção de índices.

Seguindo as fases para *self-tuning* definidas em Weikum et al.[56], durante a fase de observação o *HypoPlans* monitora e analisa cada tarefa da carga de trabalho, para identificar as estruturas de banco de dados mais adequadas e seus respectivos benefícios para determinada tarefa. Durante a fase de previsão, o *HypoPlans* tenta inferir os efeitos que resultariam da alteração de uma configuração de projeto físico atual C para uma nova configuração C', onde C' contém estruturas reais e hipotéticas. A fase de reação tem a funcionalidade de alterar fisicamente uma configuração C para a nova configuração C' gerada pelo *HypoPlans*.

Utilizando o conceito de planos de execução de consultas hipotéticas, a ideia chave é permitir que os *HypoPlans* substituam um subplano *p* pertencente ao plano de execução original *P* por um hipotético subplano *p'*. *HypoPlans* pode criar um plano hipotético *HP*, que é equivalente ao plano original *P*, mas com um custo de execução estimado mais baixo. Assim, as estruturas hipotéticas em um subplano hipotético *p'* são boas candidatas para ser fisicamente criadas. Para comparar os custos de execução de *HP* e *P*, foi desenvolvido um modelo de custo canônico (CCM) usado apenas para dar suporte ao *HypoPlans* na descoberta de atividades de ajuste de projeto físico eficientes.

Embora no trabalho seja abordado somente o problema de *self-tuning* de índices, os autores enfatizam que a abordagem pode ser facilmente aplicada a outros problemas de projeto físico, como particionamento de tabelas e visões materializadas [9].

3.4. Abordagens semiautomáticas

Considerada o meio-termo entre as abordagens *on-line* e *off-line*, as técnicas de ajuste de projeto físico semiautomáticas tentam suprir as deficiências das técnicas *off-line*, selecionando de forma *on-line* uma carga de trabalho representativa para ser utilizada no processo de recomendação de índices. Algumas ações são deixadas por conta do DBA, como o momento a ser realizada a seleção, a criação ou a manutenção de índices, ou mesmo a inserção de um parecer *(feedback)* sobre os efeitos de determinada recomendação de índices implantada, que pode ser utilizado como parâmetro em recomendações posteriores.

3.4.1. *Kaizen: A Semi-Automatic Index Advisor*

Em Jimenez et al. [8] é apresentada uma ferramenta semiautomática de sintonização de índices denominada Kaizen, que visa recomendar configurações de índices (criação e remoção) baseados na carga de trabalho.

Para monitorar a carga de trabalho e prover a seleção de índices, Kaizen usa o método de dividir para conquistar no espaço de soluções e, em seguida, recomenda alterações no projeto físico relacionado a índices.

Apesar de a ferramenta sugerir e ser capaz de criar e remover índices de forma contínua, é permitido ao DBA analisar as recomendações, além de permitir que ele escolha o momento de materializar os índices.

Outra característica da ferramenta é que o DBA pode inserir um *feedback* com o objetivo de refinar recomendações futuras. Para cada sugestão de índice ela armazena o *feedback* e preferências do DBA de duas formas:

(i) explícita: quando um DBA indica se aceita ou não determinada recomendação;

(ii) implícita: diz respeito às decisões do DBA que contrariam ações automáticas, como é o caso da remoção de um índice que tenha sido sugerido anteriormente pela ferramenta. Dessa forma, a ferramenta infere que o DBA não concordou com a sua sugestão e guarda esse conhecimento para as futuras sugestões.

3.5. Considerações finais do capítulo

Pesquisadores propuseram muitas técnicas para *tuning* automático de projeto físico nas últimas três décadas. Neste capítulo, listamos os principais trabalhos que abordam o *tuning*, sobretudo para o problema de *tuning* de índices [1], [4], [8], [10], [11], [26], [36], [62].

Com exceção de [10] e [26], a maioria dessas propostas possuem características de projeto *"what-if"*. Isto é, promovem simulações com base em configurações hipotéticas, buscando projetar otimizações possíveis a partir de índices e de outras estruturas de acesso. Além disso, estes trabalhos comumente empregam heurísticas com podas gulosas para eliminar o espaço de busca. Embora a poda gulosa torne as ferramentas viáveis, ela reduz sua utilidade ao eliminar muitos candidatos úteis.

Salienta-se que iniciativas para *tuning* automático de projeto físico para SGBDs de código aberto são relativamente novas comparadas às soluções para SGBDs comerciais. Schnaitter et al. [62], Monteiro [11] e Maier et al. [26], propuseram ferramentas com importantes contribuições para *tuning*, utilizando SGBD de código aberto. No entanto, nenhuma delas considerou as diferenças de custos de E/S entre dispositivos de armazenamento HDD e SSD.

Salienta-se que, embora o custo monetário do SGBD seja um fator importante na decisão da escolha por um SGBD de código aberto, a falta de ferramentas automatizadas apropriadas para SGBDs de código aberto, torna a operação de *tuning* mais custosa que de um SGBD comercial. Portanto, destaca-se a importância de iniciativas que contemplem o uso de SGBDs de código aberto, como realizado nesta tese, bem como trabalhos que foquem na ampliação do uso de ambientes híbridos de armazenamento, visando o aprimoramento de sua administração e o incremento de desempenho.

Capítulo 4

Método

Neste capítulo é apresentado o método proposto nesta tese, caracterizando os componentes principais do algoritmo ITLCS proposto, o fluxo de etapas para sua execução, parâmetros escolhidos e utilizados pelo algoritmo, entre outros aspectos.

Como este trabalho é direcionado a ambientes híbridos de armazenamento, foram executados experimentos (detalhes no capítulo 5) em diferentes cenários, utilizando armazenamento HDD e SSD. Além disso, foram criados cenários de testes utilizando o SGBD PostgreSQL padrão, bem como outros cenários utilizando uma versão extendida do SGBD PostgreSQL implementada nesta tese (detalhes na seção 5.1 e 5.2).

Para cada cenário de teste definido (detalhado no capítulo 5) foram realizados experimentos com as etapas que seguem a abaixo.

- Geração da população inicial de classificadores: nesta etapa, são gerados os classificadores necessários para compor a base de classificadores do ITLCS desenvolvido.
- Treinamento do ITLCS:
    - Execução de 10 iterações do ITLCS, utilizando a carga de trabalho selecionada. Para definir o número de iterações do ITLCS, foram realizados experimentos preliminares com 1, 5, 10 e 20 iterações, e baseado nos experimentos foi escolhido 10 iterações para a fase de treinamento do ITLCS.

  - Descoberta de novas regras de tuning: nesta fase é realizada uma busca por uma nova geração de classificadores utilizando o AG.
  - Recomposição da população de classificadores: após a geração de novos classificadores, estes devem ser inseridos na população em substituição a outros indivíduos.

- Execução do ITLCS, para cada cenário de teste definido.

Após a execução do ITLCS nos vários cenários de teste, foi realizada uma análise comparativa dos resultados entre os cenários, verificando o desempenho alcançado pelas ações do ITLCS em cada cenário de teste. Foram analisadas também as diferenças entre regras geradas pelo ITLCS para cada cenário, conforme detalhado no capitulo 5.

Na sequência, a seção 4.1 apresenta os componentes principais do algoritmo ITLCS desenvolvido. A seção 4.2 apresenta a estrutura dos classificadores – parte antecedente e consequente. Na seção 4.3 é apresentado o processo para geração inicial de classificadores e os parâmetros iniciais escolhidos para o ITLCS. Após, na seção 4.4 é apresentado o fluxo de etapas para execução do ITLCS. Na seção 4.5 e 4.6 são apresentadas as Heurísticas utilizada pelo ITLCS. Na sequencia na seção 4.7 um ciclo completo de iteração do ITLCS é apresentado. Na seção 4.8 é apresentado as alterações propostas para o modelo de custos do SGBD PostgreSQL. Por fim na seção 4.10 discute-se os pontos principais do capítulo.

4.1. O algoritmo ITLCS

Um SGBD, durante a sua operação em um ambiente real de aplicações, pode apresentar problemas de desempenho, que entre outras opções, podem ser resolvidos por meio de ações de *tuning* de índices. Desta forma, o algoritmo ITLCS tem como meta receber uma carga de trabalho, contendo um ou mais comandos SQL, e realizar o *tuning* de índices. Portanto, quando um usuário submeter a mesma consulta ao SGBD, ou consultas similares, estas poderão ser executadas mais rapidamente utilizando o(s) índice(s) criado(s). Para realizar o *tuning* de índices, o SC gera um conjunto de regras de inferência (classificadores), no formato SE <condição> - ENTÃO <ação>, onde cada classificador tenta mapear apenas uma situação que pode ocorrer no ambiente e, além disso, existem classificadores específicos para serem utilizados quando os dados estão em HDD e outros para SSD. Desta forma,

utilizando o conjunto de classificadores de sua base, o SC ajusta o desempenho do SGBD por meio de ações de criação ou manutenção de índices presentes no consequente de cada classificador.

Visando uma convergência ágil na tentativa de obter uma solução ótima, o ITLCS utiliza configurações hipotéticas de índices [75]. A utilização de índices hipotéticos tem como vantagem a eliminação do tempo de espera para criação de índices, além de permitir a obtenção de estimativas de custos utilizando tais índices.

4.1.1. Componentes principais do ITLCS

Como apresentado na figura 4.1 e também detalhado no capítulo 2, um SC baseado no modelo de Holland [47] interage com o ambiente (SGBD) por meio de detectores e executores e organiza-se em três componentes principais: subsistema de tratamento de regras e mensagens, subsistema de descoberta de novas regras e subsistema de atribuição de crédito. Além destes componentes, em nossa proposta também foram adicionados: repositório base, módulo de heurísticas e um módulo de monitoramento, descritos nas subseções a seguir.

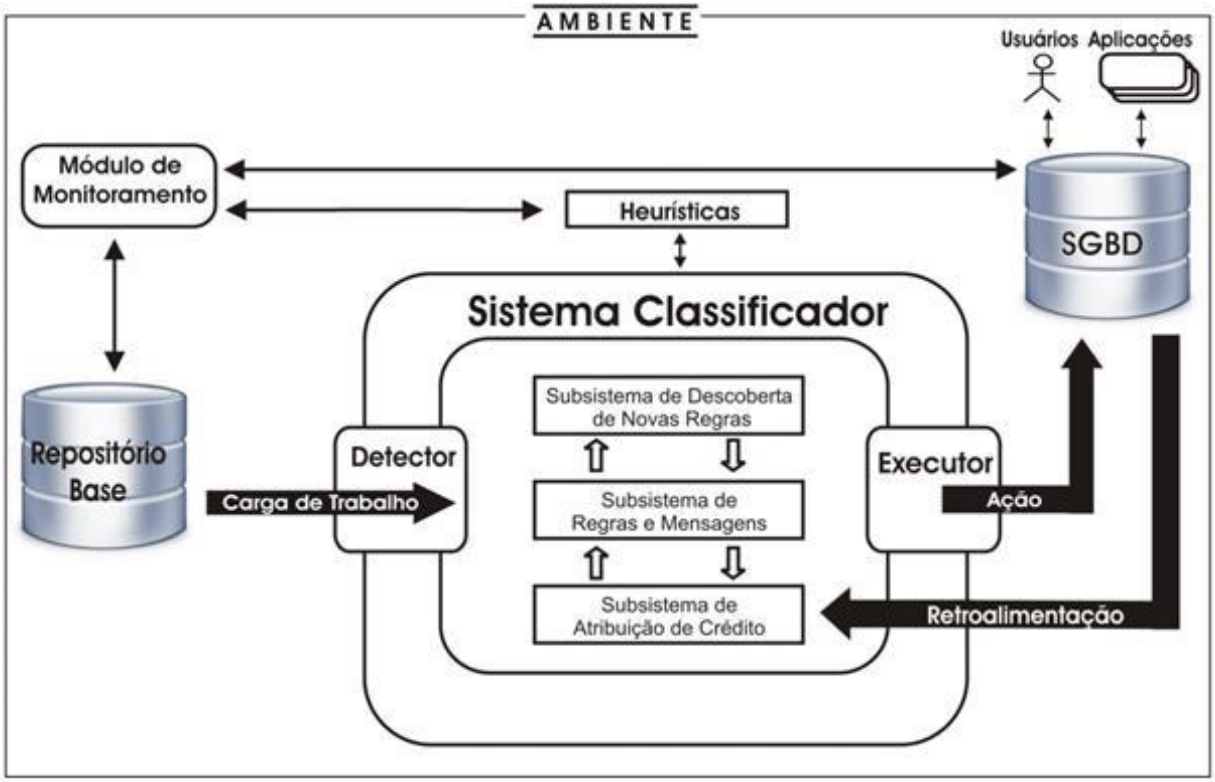


Figura 4.1. Arquitetura do sistema ITLCS
Fonte: o autor.

Repositório base

O repositório base (RB) armazena em tabelas do banco de dados os seguintes itens: a carga de trabalho capturada pelo SGBD, o plano de execução de cada comando SQL, a quantidade de tuplas retornadas e o custo calculado da consulta antes e depois de qualquer ação do SC. Destaca-se também que o RB não necessita estar fisicamente no mesmo local de armazenamento da carga de trabalho.

Módulo de monitoramento

Responsável por capturar comandos SQL da carga de trabalho, submetidos ao SGBD, o módulo de monitoramento grava no RB os comandos SQL capturados.

Na fase de captura, o módulo de monitoramento utiliza a Heurística de Monitoramento da Carga de Trabalho (HMCT), definida na seção 4.5, que foi inspirada em Sattler et al. [64]. Esta heurística utiliza o conceito de "Época", a fim de delimitar um período (parametrizado) de observação. Com isso os comandos SQL serão capturados e comporão a carga de trabalho.

Heurísticas

Como definido na seção 4.5 e 4.6, tanto a heurística de recompensa e punição (HRP) quanto a heurística HMCT, têm funções importantes. A HRP é utilizada pelo subsistema de atribuição de crédito no processo de retroalimentação do SC, visando recompensar ou punir adequadamente um classificador após a aplicação de uma ação no ambiente. A HMCT é empregada pelo módulo de monitoramento, para que seja armazenada no RB, a carga de trabalho e outras informações relevantes.

4.2. Características estruturais dos classificadores

Como mencionado anteriormente, cada indivíduo da população de classificadores é representado por uma regra de inferência do tipo SE <condição> - ENTÃO <ação> composta por dezessete campos (genes na linguagem biológica), sendo dezesseis na parte antecedente e um na parte consequente. Os dezesseis genes da parte antecedente descrevem as condições que devem ser satisfeitas para ocorrer à combinação de uma entrada com os classificadores da base de classificadores do SC. Já a parte consequente, que ocupa apenas um gene do classificador, denota a ação a ser tomada caso as condições da parte antecedente sejam verdadeiras. Os quadros 4.1 e 4.2, respectivamente, detalham o número de bits, descrição e codificação de cada gene da parte antecedente e consequente do classificador.

Quadro 4.1. Características e genes do antecedente de um classificador

Nº Gene	Nº Bits	Descrição	Codificação
G1	1	Código que indica se a tabela está armazenada em HDD ou SSD.	0 HDD 1 SSD
G2	1	Código que indica se existe índice na coluna.	0 Não 1 Sim
G3	1	Código que indica se existe alguma função de agregação na coluna.	0 Não 1 Sim
G4	1	Código que indica se a coluna é uma *Foreign Key*.	0 Não 1 Sim
G5	1	Código que indica se a coluna está em cláusulas de *Group By*.	0 Não 1 Sim
G6	1	Código que indica se a coluna está em cláusulas de *Order By*.	0 Não 1 Sim
G7	2	Código que indica qual o tipo de dado da coluna.	00 = Numérico 01 = Caracter 10 = Tempo/data 11 = Binário
G8	2	Código que indica qual operador relacional é utilizado em uma coluna.	00 (=) 01 (<,<=,>,>=) 10 In 11 Between
G9	1	Código que indica se a coluna está envolvida em um predicado de junção.	0 Não 1 Sim
G10	3	Código que indica o grau de seletividade da coluna.	000 Entre 0 e 30% 001 Entre 31 e 60% 010 Entre 61 e 70% 011 Entre 71 e 75% 100 Entre 76 e 80% 101 Entre 81 e 85% 110 Entre 86 e 90% 111 Entre 91 e 100%
G11	3	Código que indica a existência de um índice, e o seu grau de fragmentação.	000 Entre 0 e 5% 001 Entre 6 e 10% 010 Entre 11 e 20% 011 Entre 21 e 35% 100 Entre 36 e 55% 101 Entre 56 e 70% 110 Entre 71 e 85% 111 Entre 86 e 100%
G12	3	Código que indica a porcentagem de retorno de linhas na execução da consulta.	000 Entre 0 e 5% 001 Entre 6 e 10% 010 Entre 11 e 15% 011 Entre 16 e 20% 100 Entre 21 e 30% 101 Entre 31 e 40% 110 Entre 41 e 60% 111 Entre 61 e 100%
G13	3	Código que indica a porcentagem de linhas atualizadas, por comandos de *UPDATE* da carga de trabalho, referenciando a coluna.	000 Entre 0 e 5% 001 Entre 6 e 10% 010 Entre 11 e 15% 011 Entre 16 e 20% 100 Entre 21 e 30% 101 Entre 31 e 40% 110 Entre 41 e 60% 111 Entre 61 e 100%
G14	3	Código que indica a existência de um índice, e a respectiva altura da árvore B+.	000 Entre 0 e 5% 001 Entre 6 e 10% 010 Entre 11 e 15% 011 Entre 16 e 20% 100 Entre 21 e 30% 101 Entre 31 e 40% 110 Entre 41 e 60% 111 Entre 61 e 100%
G15	3	Código que indica a existência de um índice, e a razão do tamanho do índice versus o tamanho do *Shared Buffer*.	000 Entre 51 e 70% 001 Entre 71 e 85% 010 Entre 86 e 100% 011 Entre 101 e 125% 100 Entre 126 e 150% 101 Entre 151 e 175% 110 Entre 176 e 200% 111 maior que 201%
G16	3	Código que indica a razão da tabela versus memória compartilhada do SGBD.	000 Entre 51 e 70% 001 Entre 71 e 85% 010 Entre 86 e 100% 011 Entre 101 e 125% 100 Entre 126 e 150% 101 Entre 151 e 175% 110 Entre 176 e 200% 111 maior que 201%

Quadro 4.2. Características e genes do consequente de um classificador

Gene Nº	Nº de Bits	Codificação
A1	3	000 - Criar um índice em árvore B+ para uma coluna utilizando armazenamento HDD.
A2	3	001 - Criar um índice em árvore B+ para uma coluna utilizando armazenamento SSD.
A3	3	010 - Reconstruir um índice que está fragmentado.
A4	3	011 - Migrar índice existente de um HDD para um SSD.

4.2.1. Detalhamento sobre a parte antecedente do classificador

Devido ao ITLCS ser um algoritmo direcionado a ambientes híbridos de armazenamento, o gene G1 tem o propósito de identificar qual o tipo de dispositivo de armazenamento, sendo "0" para HDD e "1" para SSD. A partir dessa informação, o SC poderá realizar ações de *tuning* de forma adequada ao tipo de armazenamento.

O gene G2 foi inserido no cromossomo para identificar se o SC deve criar índice ou realizar a manutenção do mesmo, ou seja, se o valor do gene for "0" significa que não existe nenhum índice para a coluna envolvida e desta forma o SC poderia criar um índice. No entanto, se a coluna envolvida já tiver um índice, poderiam ser aplicadas apenas ações de manutenção (ações de código 010 e 011).

Os genes G3 a G8 foram criados baseados principalmente na heurística de seleção de índices proposta em Lohman et al. [10] e implementada no SGBD IBM DB2. A heurística sugere que a seleção de índices deve ser fortemente integrada ao otimizador do SGBD. A heurística analisa predicados e cláusulas presentes no comando SQL submetido para encontrar colunas que poderiam ser indexadas, como a seguir:

(i) colunas envolvidas em predicados de igualdade;
(ii) colunas envolvidas em cláusulas ORDER BY, GROUP BY e predicados de junção;
(iii) colunas que aparecem em restrições de intervalos;
(iv) colunas que aparecem em outros predicados indexáveis;
(v) demais colunas referenciadas no comando SQL.

De modo complementar, para embasar os critérios dos genes G3 a G8, seguiu-se a análise de outros autores reconhecidos da área de SGBDR como [2], [12], [13], [21], que também abordam estes critérios.

O gene G10 representa o grau de seletividade da coluna envolvida. Ele foi inserido, pois a seletividade[2] de um atributo de uma tabela representa um fator importante na decisão de criação de um índice [13] e considerando HDDs, segundo Chan e Ashdown [13], normalmente um índice é vantajoso quando o grau de seletividade da coluna é maior que 85%.

O gene G11 foi inserido no cromossomo devido à ação de reconstrução de índice, pois através deste são extraídas as entradas de informação referentes ao nível de fragmentação (caso existam índices para a coluna em questão). Como discutido no capítulo 2 – seção 2.4.3, em árvores B+ as operações de *insert, update* ou *delete*, tendem a provocar divisões de páginas no nível folha do índice *(page-splits)* e neste caso, uma das soluções possíveis, seria a reconstrução do índice, processo que deixará novamente o índice ordenado fisicamente, provendo maior desempenho em operações de varredura [71].

Para o gene G12, é extraída a quantidade retornada de linhas da consulta, que é uma informação importante para tomada de decisão, pois segundo encontrado em Chan e Ashdown [13], considerando HDDs, um índice é vantajoso para o desempenho de uma consulta caso o retorno de linhas seja menor que 15%.

Atualizações em colunas indexadas podem causar deterioração do desempenho do sistema. Desta forma, o gene G13 tem por meta, considerando uma coluna específica, verificar a quantidade de atualizações daquela coluna em outros comandos da carga de trabalho.

Em G14, caso exista um índice para coluna analisada, é extraída e armazenada a altura da árvore B+ do índice. A altura é um importante fator para inferir sobre o seu tamanho e decidir se este deve ser migrado de um dispositivo HDD para um SSD, por exemplo.

O gene G15 tem a função de extrair a razão do tamanho do índice em relação ao tamanho da área de memória *cache* compartilhada, reservada para o SGBD, que no PostgreSQL é denominada *shared buffer*. Isto é relevante, pois um índice pode ser considerado grande, caso seu tamanho seja maior que o *shared buffer*. Baseado nessa informação, o SC pode tomar melhores decisões sobre a desfragmentação ou não deste.

De forma similar o gene G16 tem a função de extrair a razão do tamanho da tabela em relação ao tamanho da memória cache compartilhada – exclusiva para uso do SGBD. Considerando, por exemplo, que uma tabela seja maior que a memória compartilhada, pode-se

[2] Um atributo com baixa seletividade significa que os dados daquele campo têm valores muito pouco distintos.

concluir que seu acesso não será totalmente em memória RAM, e neste caso a criação de um índice pode ser vantajosa, por reduzir o número de acessos a disco.

Vale destacar que caso seja necessário tratar novas condições das entradas, não previstas anteriormente, o SC proposto permite que sejam incluídos mais genes na parte antecedente do classificador.

4.2.2. Detalhamento sobre a parte consequente do classificador

Nesta subseção são discutidas as ações que foram utilizadas pelo ITLCS na fase de experimentação. No entanto, ressalta-se que outras ações podem ser inseridas, e demandam poucas alterações no algoritmo ITLCS.

Como principais ações do ITLCS temos a criação de índices B+ Tree em HDD e em SSD, que são as ações A1 e A2 respectivamente (quadro 4.2). A diferenciação entre criação de índices em HDD e SSD se faz necessária por questões de implementação do ITLCS, demandados também pelo processo de migração, pois na migração de um índice, o ITLCS invoca a ação de criação de índices em SSD.

Além da capacidade de criação de índices, o ITLCS proposto tem duas características não encontradas nas ferramentas POWA e EDB, avaliadas neste trabalho e em outras disponíveis comumente utilizadas por DBAs, que são a reconstrução e a migração de índices de um dispositivo de armazenamento para outro, detalhadas nos próximos parágrafos.

A reconstrução de índices – ação A3, é um recurso utilizado por DBAs para corrigir a fragmentação de índices, conforme abordado no capítulo 2 – seção 2.4.3. Para selecionar quais índices estão fragmentados, o ITLCS dispõe de classificadores específicos que contenham a ação de reconstrução de índices na parte consequente. Salientamos que no quadro 4.1 os genes que influenciam na decisão de reconstrução de índices são os genes 2 e 11. Desta forma, caso exista alguma entrada que combine sua parte antecedente com a parte antecedente de algum classificador presente no ITLCS e este seja selecionado, o índice será reconstruído.

Concernente à ação de migração de índice de um dispositivo A para um dispositivo mais ágil B – ação A4, destacamos que se trata de uma ação importante em ambientes híbridos de armazenamento, principalmente para índices de grandes tabelas, agilizando o acesso em operações de atualização do(s) índice(s), e permitindo que o índice migrado possa usufruir de recursos de paralelismo – implícitos aos SSDs, em acessos de leitura e escrita.

Para selecionar quais índices deverão ser migrados de um disco HDD para um SSD, o ITLCS dispõe de classificadores específicos. O quadro 4.1, mostra os genes que influenciam na decisão de migração de índices, são eles: G1, G2, G14 e G15.

4.3. População inicial de classificadores e parâmetros do algoritmo

Em geral os AGs são capazes de evoluir para soluções satisfatórias, desde que exista diversidade entre os indivíduos da população inicial [44], [83]. Além disso, quando for possível a introdução de algum conhecimento inicial, utilizando heurísticas por exemplo, o aprendizado será facilitado. Em virtude disto, neste estudo definiu-se que os classificadores iniciais serão gerados 50% de forma aleatória e 50% de forma manual.

Para a geração manual dos classificadores, foram utilizadas heurísticas presentes na literatura destinadas ao problema de seleção de índices [10], [18], [50], [75]. De modo complementar, também foram consultados livros da área de SGBD relacionais [2], [12], [21], [24], [45], [68], identificando boas práticas e técnicas amplamente utilizadas para seleção e manutenção de índices.

Destaca-se que os classificadores da população inicial são especializados somente para ambientes com armazenamento em HDD. No entanto, eles serão aplicados tanto para ambientes HDD quanto para SSD, pois, conforme se espera, o ITLCS deve adaptar automaticamente a população de classificadores ao novo ambiente de armazenamento com SSD.

Outro aspecto relevante é que a estratégia de combinar/unir vários critérios/condições para concepção de uma regra de inferência representa uma técnica similar àquela que um DBA ou um especialista humano faria caso necessitasse selecionar possíveis índices para uma tabela.

Os parâmetros que necessitam ser inicializados e os valores utilizados na fase de experimentação são mostrados na tabela 4.1.

A escolha ideal do número de indivíduos de uma população é comumente dependente do problema, mas como comentado no capitulo 2 – subseção 2.7.2, um número muito pequeno pode levar a população a perder sua diversidade. Em Das et al. [84] é sugerido que o número de indivíduos deve ser 10 vezes o número de genes por indivíduo. Como foram gerados indivíduos com 16 genes, temos 10*16 indivíduos. No entanto, como o ITLCS desenvolvido tem dois grupos de regras, onde cada grupo é aplicado em um tipo de

armazenamento, definiu-se o total de indivíduos da população como 2 * 10 * 16 = 320 classificadores.

Tabela 4.1. Parâmetros iniciais do ITLCS.

Parâmetros	**Valores**
TAM_POPULAÇÃO	320
NÚMERO DE GERAÇÕES	40
PROB_CROSSOVER	80%
PROB_MUTAÇÃO	5%
PER_POP_SUBS	10%
ELITISMO	4

Para o parametro PROB_CROSSOVER (percentual máximo para *crossover*), foi empregada a taxa de 80%, pois dessa forma a evolução do algoritmo nao será tão lenta.

Para o parâmetro PROB_MUTAÇÃO (percentual máximo para mutação), a taxa escolhida foi de 5%, visando assim evitar que o processo entre em estagnação, permitindo também que novas regiões do espaço de soluções sejam exploradas por meio do operador de mutação. Valores maiores que 5% poderiam tornar o processo aleatório, levando inclusive a perda de bons indivíduos.

Para o parâmetro PER_POP_SUBS (percentual máximo de indivíduos que serão substituidos), a taxa escolhida seguiu o mecanismo de recomposição adotado por Richards [73]. Esse mecanismo está descrito e ilustrado na subseção 4.4 - Figuras 4.6 e 4.7 .

Para o parâmetro ELITISMO, foi utilizado 4 indivíduos por geração, permitindo assim que os melhores indivíduos de cada geração sejam mantidos na geração seguinte. Tal valor foi definido empiricamente, uma vez que possibilitou a manutenção de indivíduos de alta aptidão sem levar o AG a uma convergência precoce.

Para definir o número de gerações do AG, foram realizados experimentos preliminares com 10, 40, 100 e 200 gerações, e baseado nos experimentos foi escolhido o total de 40 gerações.

4.4. Fluxo de etapas do ITLCS

Conforme a figura 4.2, a sequência de etapas para execução de um ciclo de iterações e posteriormente para início e fim do processo evolutivo, ocorre da seguinte forma:

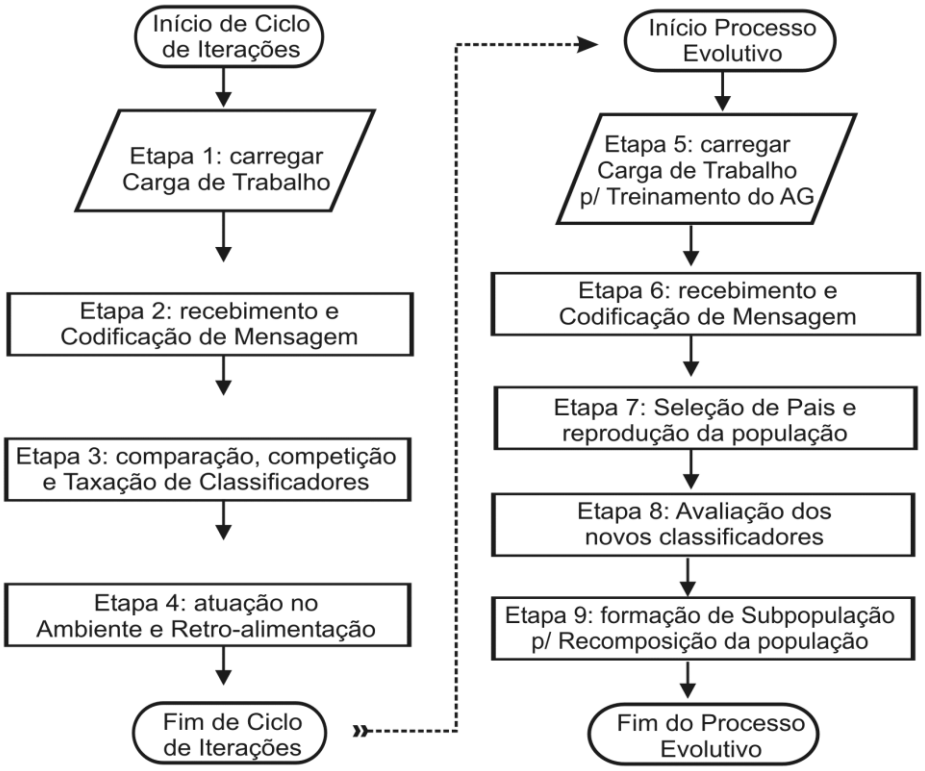


Figura 4.2. Fluxo de um ciclo de iterações e processo evolutivo.
Fonte: o autor.

1) O DBA escolhe uma janela de tempo, chamada Época, onde a carga de trabalho é capturada/carregada.
2) Após a captura da carga de trabalho, é invocado o subsistema de regras e mensagens do SC, para ser realizado o processo de codificação das entradas. Para exemplificar, considere que o SGBD recebeu como carga de trabalho somente uma consulta SQL, ilustrada na figura 4.3, onde é possível verificar que as colunas presentes na cláusula where (l_receiptdate, l_commitdate, l_partkey) foram codificadas em formato cromossômico, utilizando o alfabeto binário, contendo 16 genes (G1 a G16) conforme estrutura do classificador definido na seção 4.2. Colunas com funções Max, Sum, Avg, também podem ser codificadas, caso estejam presentes na cláusula *Select*. Ressalta-se que as entradas somente têm a parte antecedente e os classificadores têm parte antecedente e consequente.

Entrada: Consulta SQL (carga de trabalho)

```
SELECT    *
FROM      lineitem
WHERE     l_receiptdate < l_commitdate
ORDER BY  l_partkey
```

Saída: Mensagem codificada

Colunas	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12	G13	G14	G15	G16
l_receiptdate	1	0	0	0	0	0	01	01	0	001	000	010	001	110	001	110
l_commitdate	1	0	0	0	0	0	01	10	0	000	000	010	000	110	000	110
l_partkey	1	0	0	0	0	1	10	00	0	101	011	010	011	110	011	110

Figura 4.3. Exemplo de codificação de mensagem recebida.
Fonte: o autor.

3) Após a codificação das entradas, na terceira etapa todos os classificadores do SC tentam combinar sua parte antecedente com a mensagem de entrada e caso haja algum classificador que combine com a mensagem, este poderá atuar no ambiente. Na figura 4.4 é possível verificar um exemplo de classificador que consta na população inicial, bem como o significado dos genes que combinaram (*match*) com uma entrada.

Nº Gene	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**AÇÃO**
Código Binário	1	#	#	#	#	1	##	##	#	###	110	010	###	###	###	###	001

SE
G1: tabela está armazenada em disco SSD ***E***
G6: coluna é utilizada em uma cláusula ORDER BY ***E***
G10: seletividade da coluna é > 85% ***E***
G12: porcentagem de retorno de linhas é entre 11 e 15 %
ENTÃO
AÇÃO A2: criar um índice B+ tree, utilizando armazenamento SSD.

Figura 4.4. Exemplo de regra codificada em binário e a descrição de alguns genes.
Fonte: o autor.

Durante o processo de comparação, caso exista mais de um classificador que se identifique com a mensagem, elege-se um ganhador (figura 4.5), considerando o lance oferecido por cada classificador. Ao vencedor será concedido o direito de atuar sobre o ambiente.

4) Após a definição do classificador ganhador, é permitido ao mesmo executar a ação no ambiente (SGBD). Em seguida, é coletado um *feedback* do ambiente, o qual é processado pelo subsistema de atribuição de crédito do SC, para atribuir um valor de recompensa ou punição ao classificador. Para o cálculo do *feedback*, foi desenvolvida

a heurística HRPC, apresentada na seção 4.6, com base nos benefícios das ações dos classificadores. Após o recebimento do *feedback*, o sistema poderá retornar a etapa 1, ou prosseguir para a etapa 5, conforme as etapas da Figura 4.2.

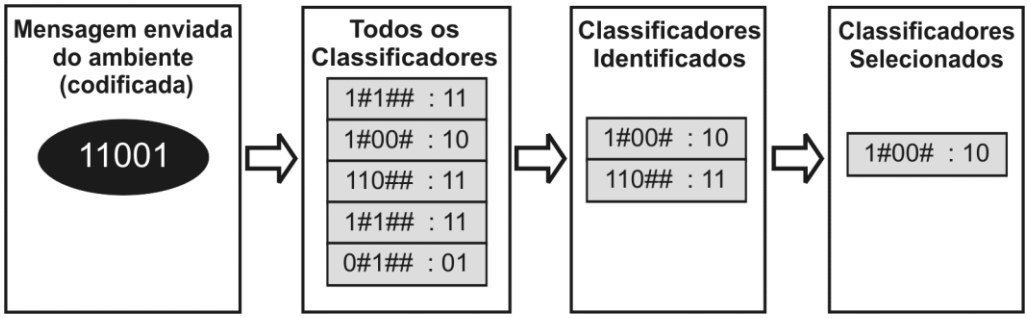


Figura 4.5. Processo de comparação de uma mensagem de entrada com os classificadores
Fonte: o autor.

Início do processo evolutivo

Caso seja escolhido iniciar o processo evolutivo (etapas 5 a 9 – Figura 4.2), por mudanças no ambiente ou mau desempenho do sistema, é invocado o subsistema de descoberta de novas regras do SC. Neste subsistema, é utilizado um AG que através de seus operadores genéticos permitem a descoberta de classificadores com desempenho aprimorado e adequado à realidade do ambiente. Para iniciar o AG é necessária a configuração dos seguintes parâmetros:

- TAM_POPULAÇÃO (nº inteiro positivo): representa o número de indivíduos da população;
- NÚMERO DE GERAÇÕES (nº inteiro positivo): representa o número de gerações no processo de evolução do AG;
- PROB_CROSSOVER (0% a 100%): representa a probabilidade de ocorrência da operação de cruzamento no sistema;
- PROB_MUTAÇÃO (0% a 100%): representa a probabilidade de ocorrência da operação de mutação no sistema;
- PER_POP_SUBS (0% a 100%): representa o percentual de indivíduos que serão substituídos na população.
- ELITISMO (nº inteiro positivo): representa o número de indivíduos de cada geração que serão mantidos na geração seguintes sem alterações.

4) Inicialmente é feito o carregamento de uma carga de trabalho, preferencialmente de tamanho maior que a utilizada na etapa 1. Depois disso, a geração da população inicial pode ser realizada.
5) Nesta etapa o subsistema de regras e mensagens do SC é invocado, para realizar o processo de codificação das entradas presentes na carga de trabalho.
6) Em seguida, é iniciada uma busca por uma nova geração de classificadores utilizando o AG, onde, inicialmente, é feita a seleção de pares de indivíduos para o processo de recombinação, utilizando a técnica de roleta (vide capítulo 2 - seção 2.7.2). Esta técnica favorece a seleção dos indivíduos mais adaptados e, naturalmente, os indivíduos com maior energia (*fitness*). No entanto, todos os indivíduos terão chances de seleção, uma vez que se pretende manter a diversidade da população.

   Para que a seleção de pares possa ocorrer com classificadores destinados à mesma ação, e também ao mesmo tipo de armazenamento, antes de ocorrer a seleção de pares para cruzamento é realizada a separação da população em nichos, de acordo com o tipo de armazenamento e tipo de ação (parte consequente), como no exemplo a seguir.

   - HDD (tipo de armazenamento)
     - Criar um índice B + tree para uma específica coluna em armazenamento HDD (ação);
     - Reconstruir um índice que está fragmentado (ação);
     - Migrar índice existente de um HDD para um SSD (ação).
   - SSD (tipo de armazenamento)
     - Criar um índice B + tree para uma específica coluna em armazenamento SSD (ação);
     - Reconstruir um índice que está fragmentado (ação).

   A separação em nichos de acordo com o tipo de armazenamento e de ação do classificador conduz os classificadores que tenham como ação a migração de índices, por exemplo, a fazer cruzamentos somente com aqueles que tenham a mesma ação. Esta estratégia prioriza a estabilidade de configurações, aplicando uma forma de reprodução aleatória, porém guiada.

   A cada dois pares de indivíduos selecionados é realizado o cruzamento, utilizando a abordagem de *crossover* de 2 pontos aleatórios, gerando assim um novo indivíduo. Adotou-se a abordagem de 2 pontos aleatórios, à medida que em testes empíricos este modo obteve os melhores resultados – se comparado as outras abordagens de cruzamento.

Após atingir o número de indivíduos gerados, equivalente àquele definido no parâmetro PER_POP_SUBS, finaliza-se o processo de recombinação. Com isso dá-se início ao processo de mutação, onde será selecionada uma pequena quantia de indivíduos – estabelecida pelo parâmetro PROB_MUTAÇÃO, para participarem do processo, utilizando a técnica de mutação simples.

7) Finalizado o processo de mutação, cada indivíduo deverá ser avaliado por meio de uma "função de avaliação". É essencial que uma função de avaliação seja representativa e diferencie na proporção correta as boas e as más soluções. Se ocorrer uma baixa precisão na avaliação, uma solução ótima pode ser descartada durante a execução do AG, além de desperdiçar tempo buscando soluções pouco promissoras. Para avaliar cada indivíduo gerado, remetem-se as etapas 3 e 4 da figura 4.2, onde é utilizado o subsistema de regras e mensagens para realizar o processo de comparação e competição entre os classificadores; em seguida são executadas as ações que os classificadores selecionados propõem, e por fim os classificadores são recompensados ou punidos. Como se pode observar, a função de avaliação utilizada pelo AG é equivalente à forma de avaliar as ações do ITLCS em um ciclo de iterações, e isto é realizado para cada nova geração de indivíduos do AG.

9) Após a geração e a avaliação dos novos indivíduos (classificadores), estes devem ser inseridos na população. Mas antes é necessário haver a formação da subpopulação de classificadores antigos que serão substituídos pelos novos. Para isso, utilizamos um mecanismo definido em Dejong [30], por meio de uma técnica de aglomeração entre uma subpopulação de baixa energia, onde o "fator de aglomeração" define qual o classificador será substituído, conforme ilustrado nas figuras 4.6 e 4.7. Neste trabalho, o tamanho da subpopulação é fixado pelo parâmetro PER_POP_SUBS, definido nas configurações iniciais do ITLCS, que representa o tamanho percentual da população para cada tipo de ação. Para a formação de subpopulações deve-se primeiramente separar toda a população de classificadores em nichos de acordo com o tipo de armazenamento (HDD/SSD) e tipo de ação (parte consequente), como realizado na etapa 6. No processo de recomposição da população, seguimos o mecanismo adotado por Richards [73] que se divide em duas fases sendo: (1) formação de subpopulação de classificadores a serem substituídos; e (2) inserção de novos classificadores na população. Conforme ilustrado na figura 4.6, a subpopulação é formada utilizando a

seleção por torneio, a partir de seleções aleatórias dentro da população, que obedecem a um "fator de reposição". Inicialmente é determinado o tamanho da subpopulação que será substituída (definida pelo parâmetro PER_POP_SUBS). Em seguida, começa um processo iterativo pelo qual três indivíduos são escolhidos aleatoriamente. Deste grupo de indivíduos selecionamos o que tem a menor energia e o introduzimos na subpopulação de indivíduos a serem substituídos. Desta forma, espera-se formar uma subpopulação com os indivíduos menos aptos à sobrevivência, ou seja, com baixa energia. O processo continua até a formação completa da subpopulação de indivíduos a serem substituídos.

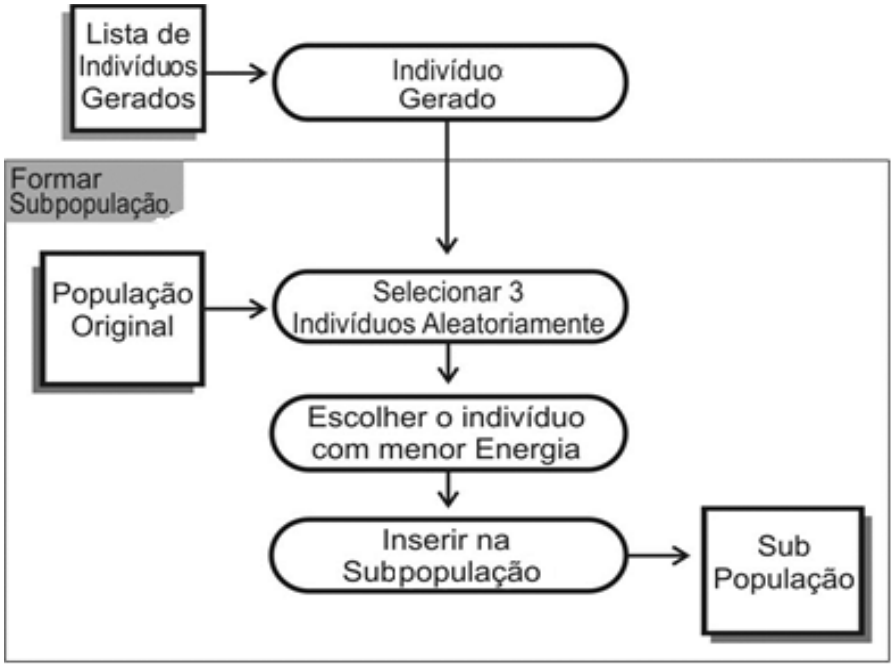


Figura 4.6. Fase I, formar subpopulação de indivíduos a serem substituídos
Fonte: o autor.

Na Fase II, responsável pela inserção dos novos classificadores na população, procede-se conforme ilustrado na Figura 4.7 onde, para cada novo classificador gerado, procura-se na subpopulação obtida na Fase I o classificador com maior "afinidade" para excluí-lo e então inserir o novo classificador. Este processo é repetido até que não se tenha mais classificadores a serem inseridos.

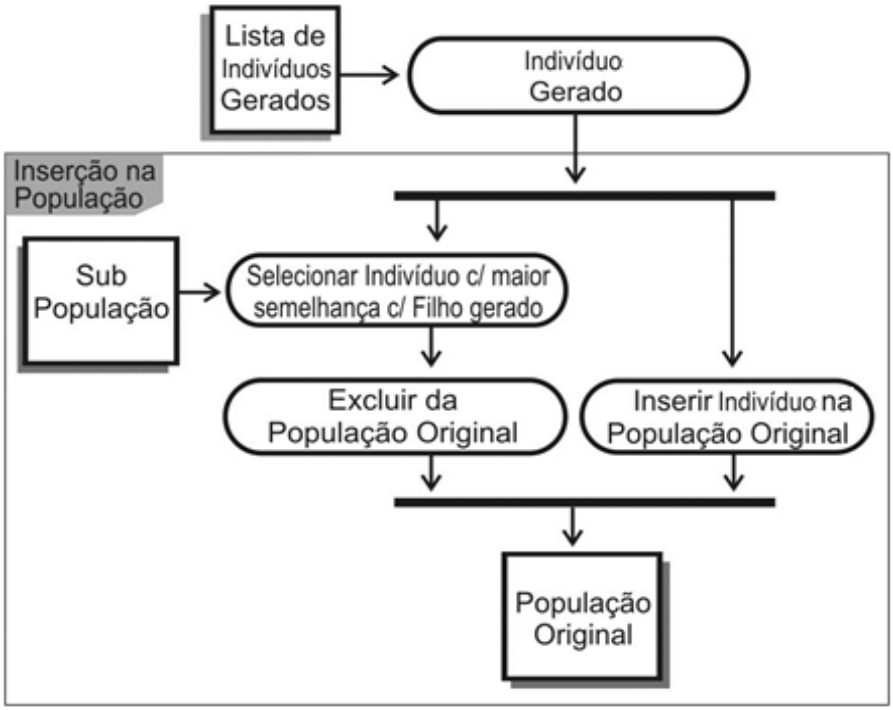


Figura 4.7. Processo de recomposição, inserção de novos classificadores na população.
Fonte: o autor.

No quadro 4.3, é possível visualizar um pseudo-código do algoritmo ITLCS, contendo de forma simplificada todo o fluxo de etapas do ITLCS, contemplando: o processo iterativo inicial; processo evolutivo para descoberta de novos classificadores; e o processo de recomposição da população.

Quadro 4.3 - Algoritmo descrevendo uma época de iterações, processo evolutivo e recomposição da população.

```
INÍCIO DO ITLCS
  INÍCIO DE UMA ÉPOCA DE ITERAÇÕES
    Passo 1: Recebimento de carga de trabalho do SGBD (mensagens).
    Passo 2: Codificação das mensagens.
    Passo 3: Selecionar classificadores ativados (matched).
    Passo 4: Início de competição:
             • Calcular o bid de cada competidor.
             • Selecionar classificador vencedor.
             • Aplicar taxas para competidores e para o vencedor.
    Passo 5: Aplicar ações no SGBD (ambiente).
    Passo 6: Receber feedback do ambiente.
    Passo 7: Recompensar e punir classificadores.
    Passo 8: Aplicar taxa de vida a todos os classificadores.
  FIM DE UMA ÉPOCA
  INÍCIO DO PROCESSO EVOLUTIVO (executar AG)
    Passo 9: Recebimento da uma carga de trabalho do SGBD (mensagem).
    Passo 10: Codificação das mensagens.
    Passo 11: Início de uma nova geração de classificadores:
             • Selecionar pares de classificadores para aplicação de operadores genéticos.
             • Aplicar o operador de crossover para geração dos filhos.
             • Aplicar o operador de mutação em parte dos filhos.
             • Avaliar os indivíduos gerados, utilizando uma função de avaliação.
    Passo 12: Senão é o limite de gerações, ir para passo 11.
  FIM DO PROCESSO EVOLUTIVO
  INÍCIO DO PROCESSO DE RECOMPOSIÇÃO DE CLASSIFICADORES
    Passo 13: Selecionar uma subpopulação de classificadores a serem substituidos.
    Passo 14: Substitua os classificadores selecionados na Etapa 13 pelos
              classificadores gerados ao final da Etapa 11.
  FIM DO PROCESSO DE RECOMPOSIÇÃO DE CLASSIFICADORES
END.
```

4.5. Heurística de monitoramento da carga de trabalho (HMCT)

Inspirada em Sattler et al. [64] a HMCT utiliza o conceito de "época", que basicamente consiste em delimitar um período de observação para que os comandos SQL sejam capturados e assim compor a carga de trabalho que posteriormente será submetida ao SC. Neste trabalho, uma "época" pode ser definida/parametrizada com as opções seguintes.

- Manual: comandos SQL DML podem ser inseridos manualmente para ser submetido ao SC.
- Tempo: estabelece o tempo (em minutos) para captura de comandos SQL. Finalizado esse tempo a carga de trabalho capturada será submetida ao SC.
- Número de comandos SQL: será estabelecido um número de comandos que serão capturados e incluídos na carga de trabalho.

Além da possibilidade de parametrização da janela de captura da carga de trabalho, a utilização da HMCT trata dois problemas potenciais: (i) a desatualização das estatísticas e (ii) a escolha do momento correto para iniciar a etapa de predição de índices. Na busca por melhoria de desempenho, estatísticas desatualizadas podem influenciar negativamente levando a criação de estruturas desnecessárias. É importante também a escolha do momento adequado para se iniciar a fase de predição, pois fazê-la toda vez que uma consulta for submetida ao SGBD, conforme [27], [57], [64], [66] e [71], poderá degradar o desempenho.

4.6. Heurística de recompensa e punição de classificadores (HRPC)

Esta seção tem por objetivo detalhar a heurística HRPC, utilizada pelo processo de retroalimentação do SC, quando alguma ação é executada no ambiente.

4.6.1. Benefícios de estruturas

A avaliação de estruturas de acesso utilizando como base o(s) benefício(s) proporcionado(s) pelo seu uso, como índices e visões materializadas, já foi utilizada em vários trabalhos como em [3], [4], [41], [66] e [71].

Considerando o uso de índices, a ideia básica é que o benefício seja qualificado na medida em que o índice esteja envolvido em comandos executados pelo SGBD. O benefício

deve ser incrementado caso auxilie na diminuição do tempo de execução ou na redução de custos, e decrementado caso ocorra o contrário.

Formalmente, pode-se definir o benefício Bej, qk proporcionado por uma estrutura ej para executar uma tarefa qk, como:

$$B_{ej,qk} = (EC(qk) - EC(qk, ej)) \quad \text{(Exp. 4.1)}$$

onde, EC(qk) é a estimativa de custo de execução da tarefa qk sem utilizar a estrutura ej;

e EC(qk, ej) é a estimativa de custo da tarefa qk utilizando-se da estrutura de acesso ej.

Se EC (qk) − EC (qk, ej) > 0, então a estrutura ej contribui para realização da tarefa qk, melhorando seu desempenho. De outro modo, se EC(qk) − EC(qk, ej) < 0, a estrutura ej não contribui para o aumento de desempenho da tarefa qk. No entanto, vale destacar que mesmo que uma estrutura ej seja criada no SGBD e não contribua para o desempenho da tarefa, o otimizador de consultas poderá simplesmente ignorá-la e não utiliza-lá.

4.6.2. Benefício percentual

Para facilitar a análise do benefício de uma estrutura pode-se estimá-la de forma percentual como:

$$BP_{ej} = 1 - \left(\frac{EC(qk,ej)}{EC(qk)}\right) \quad \text{(Exp. 4.2)}$$

Onde o custo (qk, ej) é o custo de execução, considerando uma estrutura de acesso (índice, visão materializada); e o custo (qk) é o custo sem a estrutura de acesso.

Exemplo de aplicação:

Considere a consulta[3] apresentada no quadro 4.4:

Quadro 4.4. Comando SQL

```
Select o_orderdate, l_shipdate
from lineitem l, orders o
where
        o.o_orderkey = l.l_orderkey
         and l.l_supkey = 50
```

Observe também um plano de consultas para esta mesma consulta, ilustrado na figura 4.8.

[3] As tabelas referenciadas pertencem ao *benchmark* TPC-H [82] e os planos de execução foram extraídos do PostgreSQL.

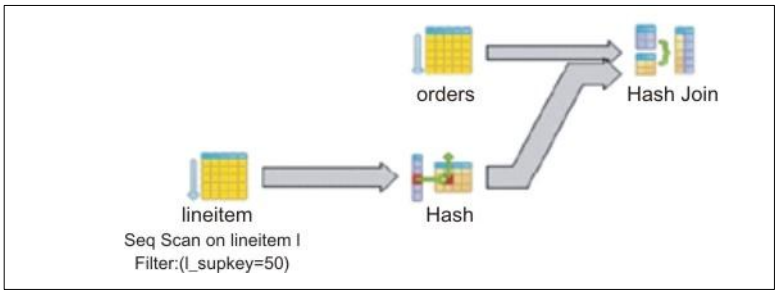


Figura 4.8. Plano de execução gráfico para a consulta do quadro 4.4
Fonte: o autor.

Considerando que o ambiente de armazenamento secundário seja HDD e que nenhum índice tenha sido criado para as tabelas envolvidas na consulta do quadro 4.4, a estimativa de custo $EC_{(SeqScan)}$ pode ser calculada por meio do comando <*explain*> do SGBD PostgreSQL.

Agora suponhamos que uma regra selecionada pelo SC tenha como ação a criação de um índice secundário para o atributo "l_supkey" da tabela lineitem. Neste caso, o subplano p composto pela operação "SeqScan on lineitem" (figura 4.8) poderia ser substituído por um subplano alternativo p' composto por uma operação "*Index Scan*" sobre a coluna l_supkey, conforme plano gerado na figura 4.9.

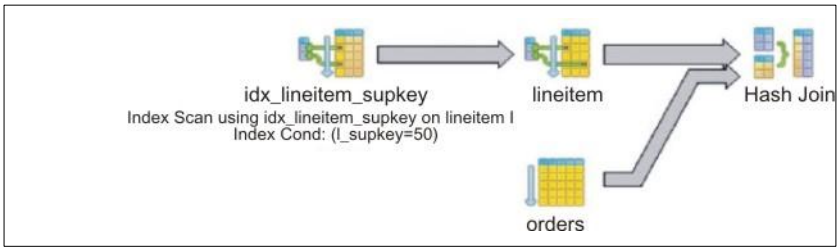


Figura 4.9. Plano alternativo gerado para a consulta do quadro 4.4
Fonte: o autor.

Para o cálculo do custo da estimativa para o subplano alternativo p', novamente utiliza-se o comando <*explain*> do SGBD PostgreSQL.

Para inferir sobre a criação ou não de um índice, é realizada a comparação dos custos de $EC_{(SeqScan)}$ com $EC_{(IndexScan)}$, realizando a subtração entre o custo sem a estrutura e o custo com a estrutura, como:

$$B_{ej,qk} = \{EC\ (qk) - EC\ (qk,ej)\}$$

Depois, com os resultados obtidos pode-se calcular o benefício percentual $BP_{ej,qk,}$ conforme expressão 4.2 definida anteriormente.

$$BP_{ej} = 1 - \left(\frac{EC(qk,ej)}{EC(qk)}\right)$$

Neste exemplo de aplicação, considerando que o índice secundário sobre o atributo l_supkey proporciona um custo menor do que uma varredura sequencial, logo, $BP_{ej} > 0$, sendo a alternativa mais vantajosa para este exemplo.

4.6.3. Custo de atualizações

Operações de atualizações com os comandos *"update", "insert"* e *"delete"* necessitam ser tratadas, pois caso uma tabela *t* necessite ser atualizada, os índices podem causar uma sobrecarga extra, em virtude de atualizações dos índices vinculados à tabela, gerando um malefício.

No entanto, apesar do custo de atualização do índice, sua presença pode acelerar o desempenho da operação, pois em atualizações, assim como em consultas, o SGBD deve inicialmente trazer os dados para a memória, para então proceder as modificações nesses dados.

Devido aos SGBDs normalmente não disponibilizarem estimativas de atualização nos seus metadados, em Salles [66] é proposta a expressão 4.3 para estimar o custo de atualização de um índice, quando utilizando um HDD:

$$\mathrm{ECA_{HDD}} = 2[\frac{\|R\|_a}{\|R\|_t}]\|R\|_p + c\|R\|_a \qquad \text{(Exp. 4.3)}$$

Na fórmula (expressão 4.3), $\|R\|_a$ é o número de registros atualizados pelo comando; $\|R\|_t$ é o número de registros da tabela; $\|R\|_p$ é o número de páginas da tabela e c é um coeficiente que relaciona, percentualmente, o custo de uma operação de E/S com o custo de CPU para processar um registro que esteja em memória, normalmente estipulado em 1%.

No entanto, devido aos custos de E/S dos SSDs serem menores do que em HDDs, adaptamos a fórmula proposta por Salles [66] para ser utilizada em SSD, conforme pode ser visto na expressão 4.4. Desta forma, a proposta é que seja reduzido pela metade o valor do primeiro coeficiente (2) e do último coeficiente (c), considerando o pior caso.

$$\mathrm{ECA_{SSD}} = [\frac{\|R\|_a}{\|R\|_t}]\|R\|_p + c\|R\|_a \qquad \text{(Exp. 4.4)}$$

Portanto, quando um índice for afetado por atualizações o seu benefício será decrementado, subtraindo o valor estimado $\mathrm{ECA_{Ai}}$. Conforme quadro 4.5.

Quadro 4.5. Algoritmo para avaliação de atualizações

```
Obter_Carga_Trabalho(Lista_Comandos_SQL)
Obter_Índices_Reais(Lista_Índices)
PARA cada item de Lista_Comandos_SQL  FAÇA
        PARA cada item Lista_Índices FAÇA
            SE (índice é afetado por Atualização) ENTÃO
                    SE (HDD) ENTÃO
                            Bej ← Bej - ECAHDD
                    SENÃO
                            Bej ← Bej - ECASSD
                     FIM SE
                FIM SE
            FIM PARA
    FIM PARA
```

4.6.4. Custo de criação (CC)

Um dos pontos de atenção em ações de *tuning* de índices é o custo de criação de um índice, pois em tabelas com grande volume de dados, pode-se levar até mesmo horas para finalizar a criação de um índice. Em Salles [66] é proposta a expressão 4.5, que faz uma estimativa de custo da criação de uma estrutura de índice, mas somente considerando HDDs.

$$CCI_{HDD} = 2P + cRlogR \qquad \text{(Exp. 4.5)}$$

A política de criação de índices se baseia em considerar que todas as páginas da tabela são lidas, ordenadas e então o índice é criado de uma forma *bottom-up*. O primeiro termo da fórmula leva em conta o custo de E/S de ler todas as páginas da tabela e de escrever todas as páginas do índice. Já o segundo termo estima o custo de ordenar todas as linhas da tabela em memória.

No entanto, considerando o uso de armazenamento SSD, neste trabalho é proposta uma nova fórmula (Exp. 4.6) para o cálculo do custo de criação de índice, onde foi substituído o valor 2 para 1, pois no pior caso, considera-se que o custo de E/S para ler todas as páginas da tabela e escrever todas as páginas do índice, será reduzido pela metade usando SSD.

$$CCI_{SSD} = P + cRlogR \qquad \text{(Exp. 4.6)}$$

4.6.5. Abrangência (AB)

Para recompensar ações que tenham larga utilização na carga de trabalho, e consequentemente beneficiar classificadores mais genéricos, foi criada a métrica de abrangência, cujos valores variam de 0 a 1, onde 0 significa baixa e 1 a máxima abrangência, como definido na expressão 4.7:

$$AB = NC_e \ / \ NC_{wt} \qquad \text{(Exp. 4.7)}$$

Sendo que: NC_e é o número de comandos SQLs em que um índice foi utilizado e NC_{wt} é o número de comandos SQL da carga de trabalho. Isso significa que quanto maior o número de comandos SQL que um índice possa ser utilizado, maior será sua medida de abrangência.

4.6.6. Utilidade (UT)

A criação de uma estrutura de acesso (como índices e visões materializadas) não garante que o otimizador irá utilizá-la. Devido a isso, a métrica UT tem por objetivo indicar se um índice aparece no plano de consulta ou não. No quadro 4.6 são apresentados os valores de recompensa e punição para métrica UT.

Quadro 4.6. Descrição e valores para métrica UT.

UT = 1 → índice utilizado no plano de execução de um comando SQL;
UT = -1 → índice não utilizado no plano de execução de um comando SQL.

4.6.7. Atribuição de recompensa e punição

Nas subseções 4.6.3 a 4.6.6 foram apresentadas as métricas para avaliação de custos, benefícios e qualidade das estruturas de acessos (índices). São elas: benefício percentual (BP), custo de atualizações (ECA), abrangência (AB), utilidade (UT) e custo de criação (CC). Todas essas métricas são importantes para o processo de decisão acerca dos valores de recompensa ou da punição do classificador que determinou a ação aplicada.

Para definir se um determinado classificador (regra) deve ser recompensado ou punido são realizadas verificações, como demonstrado no quadro 4.7, onde primeiramente é realizada a verificação se BP > 0 E UT = 1. Esta verificação evidencia que a ação tomada trouxe benefícios à carga de trabalho, pois seu benefício é positivo e a estrutura está presente no plano de execução (UT=1). No entanto, deve-se considerar que utilizando SSDs a presença no plano de execução não é garantida, mesmo quando o índice é benéfico, pois a maioria dos SGBDs não consideram os custos de E/S de SSDs. Devido a isso, neste trabalho, para fins de comparação, dois SGBDs são utilizados, um padrão e outro estendido – contendo alterações no modelo de custo do SGBD para atuação em ambientes híbridos de armazenamento (detalhado no capítulo 5).

Quadro 4.7. Estrutura de decisão sobre recompensa ou punição.

```
Obter_Tarefa_SC (Lista_Ações)
 PARA cada ação da Lista_Ações FAÇA
        SE (BP > 0 E UT = 1) ENTÃO
                Obter_Recompensa();
        SENÃO
                Obter_Punição();
        FIM SE
FIM PARA
```

Recompensa

Ao ser definido que um classificador será recompensado pela sua ação (quadro 4.7) passa-se então para a etapa de atribuição do valor de recompensa. Este valor nomeado como RC é composto pelo somatório de duas métricas ponderadas definidas nas subseções 4.6.3 a 4.6.6. Cada métrica tem o valor entre 0 a 1. No entanto, conforme expressão 4.8 os valores são multiplicados por pesos, de acordo com a relevância para o processo de recompensa.

Após testes empíricos, foi definido o peso 8 para BP e peso 2 para AB. Como o objetivo principal é premiar o ganho de desempenho obtido pelo classificador, o peso maior foi atribuído a BP.

$$RC = (BP * 8 + AB * 2) / 10 \qquad \text{(Exp. 4.8)}$$

Punição

Ao ser definido que um classificador será punido (quadro 4.7), o cálculo para o valor de punição nomeado como PN - definido na expressão 4.9, é constituído pela métrica benefício percentual (BP) definida na seção 4.6.2, que no caso da punição é um valor negativo.

$$PN = (BP) \qquad \text{(Exp. 4.9)}$$

4.7. Um ciclo completo de iteração e evolução

Nesta seção, para visualização de todo o processo, serão descritas passo a passo, todas as etapas presentes no fluxograma ilustrado na figura 4.2.

Para efeito de demonstração, supõe-se que o processo evolutivo – descoberta de novas regras – será logo após um ciclo de iteração.

As etapas que compõem o fluxo do processo (figura 4.2) consistem em:

a) Etapa 1: carregar carga de trabalho

Nesta fase, o sistema recebe uma carga de trabalho que é composta por N comandos SQL, capturados pelo módulo de monitoramento durante um período de observação. No exemplo do quadro 4.8, a carga de trabalho contém apenas uma consulta SQL.

b) Etapa 2: recebimento e codificação de mensagem

Considerando que somente o comando SQL, apresentado no quadro 4.8 foi recebido durante o período de monitoramento e captura da carga de trabalho, ilustra-se a codificação da carga de trabalho recebida no quadro 4.9.

Quadro 4.8. Carga de trabalho submetida ao SC.

```
Comando SQL:
    SELECT *
    FROM lineitem
    WHERE l_receiptdate < l_commitdate
    ORDER BY l_partkey;
```

Quadro 4.9. Codificação correspondente à mensagem enviada pelo ambiente.

COLUNAS (Cláusula *Where*)	**Genes**															
	G1	**G2**	**G3**	**G4**	**G5**	**G6**	**G7**	**G8**	**G9**	**G10**	**G11**	**G12**	**G13**	**G14**	**G15**	**G16**
l_receiptdate	1	0	0	0	0	0	01	01	0	001	000	010	001	110	001	110
l_commitdate	1	0	0	0	0	0	01	01	0	000	000	010	000	110	000	110
l_partkey	1	0	0	0	0	1	00	00	0	101	011	010	011	110	011	110

c) Etapa 3: comparação, competição e taxação de classificadores

Tomando-se como pressuposto que cada cromossomo será comparado com todos os elementos da população de classificadores, na figura 4.10 o cromossomo referente à coluna "valor" foi comparado com todos os classificadores da população. Como resultado do processo de comparação, os classificadores que se identificaram com a mensagem foram os classificadores 1 e 3.

Para decidir qual dentre os classificadores 1 e 3 irá atuar no sistema, inicia-se o processo de competição, que envolve o cálculo dos seguintes itens: aposta, taxa de aposta, taxa de vida e retroalimentação (recompensa ou punição).

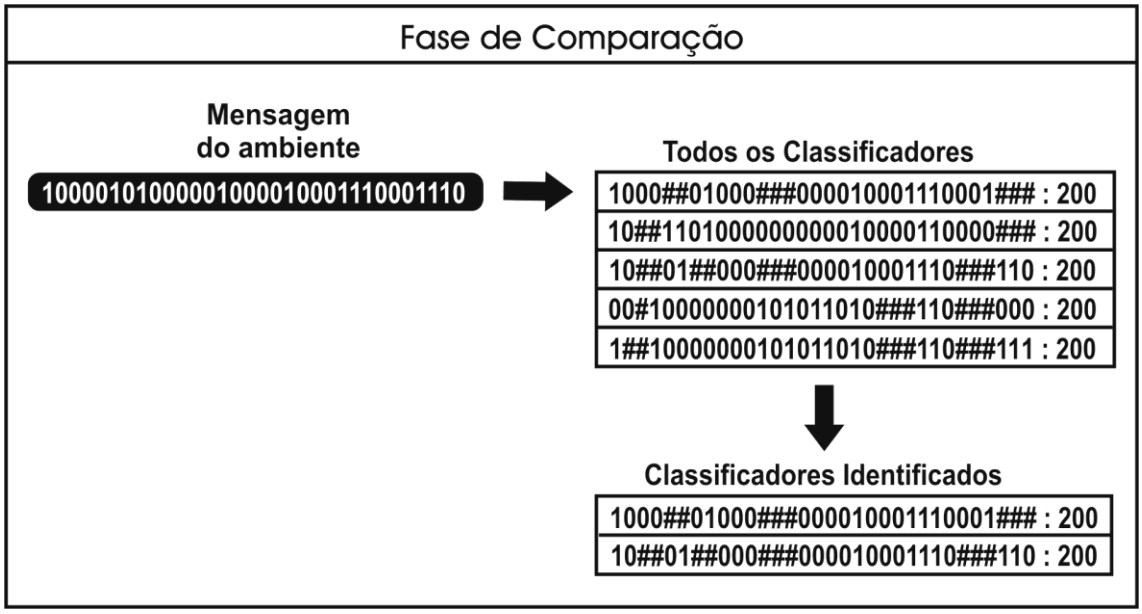


Figura 4.10. Fase de comparação e seleção de classificadores.
Fonte: o autor.

c.1) Cálculo da aposta

Considerando que os classificadores 1 e 3 combinaram com a mensagem vinda do ambiente, para calcular a sua aposta é preciso aferir sua especificidade utilizando a expressão 4.10.

$$S = (na - nt)/na \qquad \text{(Exp. 4.10)}$$

onde:

na é o comprimento do antecedente;

nt é o número de # do antecedente.

Sejam S1 e S3 as especificidades desses classificadores:

$S1 = (16 - 4) / 16 = 0{,}75$

$S3 = (16 - 9) / 16 = 0{,}43$

Utilizando a expressão 4.11, é definido o ganhador, ou seja, o classificador com maior aposta.

$$B = k0*(k1+ k2*S*k3)*E \qquad \text{(Exp. 4.11)}$$

onde:

B: aposta do classificador;

k0: coeficiente de aposta, valor positivo menor ou igual a 1;

k1: valor positivo, menor ou igual a 1, que corresponde à participação da parte não referente à especificidade na aposta;

k2: valor positivo, menor ou igual a 1, correspondente à participação da especificidade na aposta;

k3: parâmetro controlando a importância da especificidade para determinar a aposta (padrão = 1);

S: especificidade;

E: energia (aptidão) do classificador.

Quadro 4.10. Valores adotados para expressão 4.11.

k0	0,1
k1	0
k2	1
k3	1

Julgando-se que os valores adotados para a expressão 4.11 são conforme o quadro 4.10, calculando as apostas de B1 e B3 tem-se:

B1 = 0,1*S1*F1 = 0,1*0,75*200 = 15,0

B3 = 0,1*S3*F3 = 0,1*0,43*200 = 8,6

Resultando como ganhador o primeiro classificador, que obteve 15,0 de aposta.

c.2) Taxa de aposta

O classificador vitorioso é aquele que oferece maior aposta, e este valor é integralmente subtraído de sua energia. Os outros classificadores que participaram da competição multiplicam suas apostas pela taxa de aposta, TaxaBid, antes de fazerem essa subtração. Assim, a energia dos classificadores após a aposta é calculada pelas expressões 4.12 e 4.13:

$E(t+1) = E(t) - B$ para o classificador ganhador (Exp. 4.12)

$E(t+1) = E(t) - TaxaBid*B$ para os outros apostadores (Exp. 4.13)

onde:

TaxaBid: taxa de aposta

t iteração em processamento

Adotando TaxaBid = 0,8 os valores de energia do classificador ganhador e do outro apostador é dado respectivamente por:

E(t+1) = 200 – 15,0 = 185,0

E(t+1) = 200 – 0,8 * 8,6 = 193,12

c.3) Taxa de vida

Finalmente, em cada iteração, todos os classificadores ficam sujeitos a um decréscimo em sua energia devido a uma taxa de vida determinada pela expressão 4.14.

$$TxVida = 1 - (1/2)^{(1/n)} \qquad \text{(Exp. 4.14)}$$

onde:

n: vida média, medida em número de iterações.

Supondo uma vida média n = 100, obtêm-se: TxVida = 0,0069 e, utilizando este valor na expressão 4.15 aplicada a todos os classificadores, obtemos o novo valor para a energia, que pode ser visualizado no quadro 4.11:

$$E = (1 - TxVida) * E(t + 1) \qquad \text{(Exp. 4.15)}$$

Quadro 4.11. Comparativo de energia, antes e após a cobrança de taxa.

Classificador	Energia	Energia após cobrança de Taxa
1000##01000###000010001110001###: 001	200	183,72
10##11010000000000100001100000###: 001	200	198,62
10##01##000###000010001110###110: 001	200	191,78
00#10000000101011010###110###000: 001	200	198,62
1##10000000101011010###110###111: 001	200	198,62

d) Etapa 4: atuação no ambiente e retroalimentação

Neste ponto a ação do classificador é aplicada ao ambiente e um *feedback* (retroalimentação) é coletado. Considerando que o classificador 1 teve um efeito positivo no ambiente e o valor de recompensa obtido (R), por meio da heurística de atribuição de recompensa e punição foi de 3,3, então a recompensa dada será calculada, e o novo valor de energia pode ser visualizado no quadro 4.12.

E = E + B + R

E = 183,72 + 15,2 + 3,3

E = 202,02

Quadro 4.12. Comparativo de energia, antes e após a etapa de retroalimentação.

Classificador	Energia	Energia após cobrança de Taxa	Energia após Retroalimentação
1000##01000###000010001110001###: 001	**200**	**183,52**	**202,22**
10##11010000000000100001100000###: 001	200	198,62	198,62
10##01##000###000010001110###110: 001	200	191,78	191,78
00#100000001010110l0###110###000: 001	200	198,62	198,62
1##100000001010110l0###110###111: 001	200	198,62	198,62

Processo evolutivo – descoberta de novas regras

e) Etapa 5: carregar carga de trabalho para treinamento do AG

Nesta etapa é desejável que a carga de trabalho de entrada contemple mais comandos SQL do que os utilizados durante uma operação normal do SC, provendo assim um melhor ambiente para treinamento do modelo utilizando o AG.

f) Etapa 6: seleção de pais, recombinação e mutação

Consiste na seleção dos pais para posterior recombinação, utilizando-se a técnica de roleta (definida na seção 2.7.2). Após esta seleção é aplicada à recombinação (*crossover*) de dois pontos, conforme ilustrado na figura 4.11.

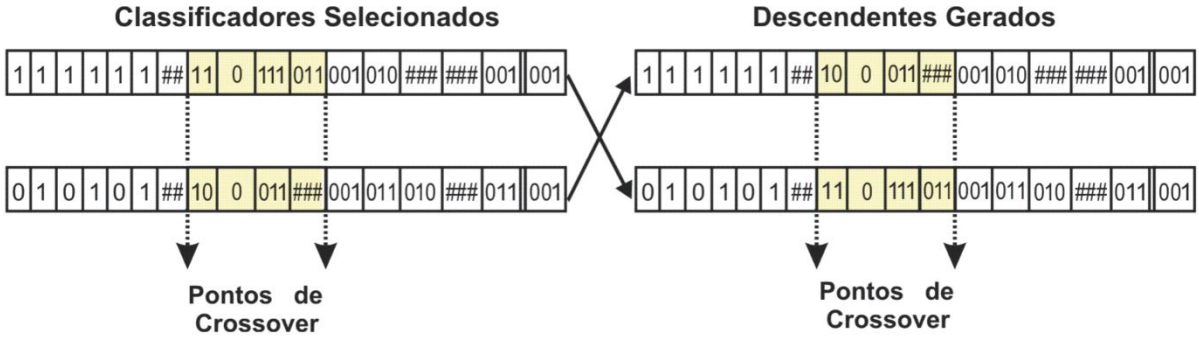


Figura 4.11. Exemplo de *crossover* de dois pontos.
Fonte: o autor.

Após a operação de *crossover*, inicia-se a operação de mutação. Para ilustrar esse processo, na figura 4.12 o gene de locus nº 6 – escolhido aleatoriamente – é mutado, ou seja, o símbolo "1', passo a ter o valor "0".

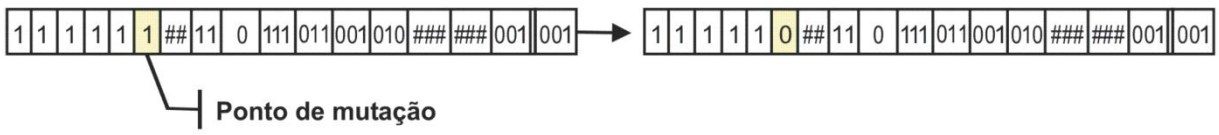


Figura 4.12. Exemplo de mutação simples.
Fonte: o autor.

g) Etapa 7: avaliação dos descendentes, novos classificadores

Neste estágio do fluxo, os descendentes gerados são avaliados com a base de treinamento. Ao final do processo será atribuído um valor de aptidão.

h) Etapa 8: formação de subpopulação para substituição de classificadores

Neste passo serão selecionados dentre a população de classificadores, indivíduos destinados a serem substituídos. Para isso é realizado o processo de torneio, onde aqueles com menor energia terão maior probabilidade de serem selecionados.

i) Etapa 9: recomposição da população com novos classificadores

Neste passo, ilustrado na figura 4.13, para cada novo indivíduo (classificador) gerado, procura-se na subpopulação obtida na etapa 8 (anterior) o classificador com maior "afinidade" para excluí-lo e então inserir o novo classificador. Este processo é repetido até que não se tenha mais classificadores a serem inseridos.

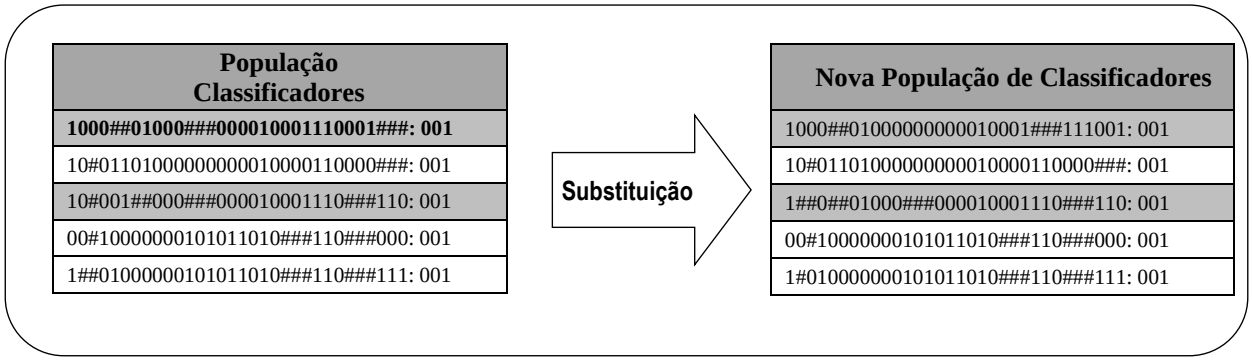


Figura 4.13. Processo de substituição da população de classificadores.

4.8. Extenções no modelo de custos do SGBD PostgreSQL

No SGBD PostgreSQL os dois principais padrões de acesso de E/S são relativos aos custos de acesso sequencial ou aleatório a uma página do disco, e são respectivamente representados pelos parametros seq_pag_cost e random_page_cost, os quais podem ser configurados por meio de arquivos de sistema.

Quando uma consulta é submetida ao otimizador do PostgreSQL, este explora vários planos na tentativa de encontrar o plano ótimo usando seu modelo de custo interno. Em linhas gerais, o PostgreSQL utiliza em seu modelo de custo um vetor de cinco parâmetros chamados de unidades de custo, a saber: $c = (c_s, c_r, c_t, c_i, c_o)^T$

Onde:

- c_s (seq page cost): custo para processar sequencialmente uma página no disco;
- c_r (random page cost): custo para buscar aleatoriamente uma página no disco;
- c_t (cpu tuple cost): custo de processamento de UCP para uma tupla;
- c_i (cpu index tuple cost): custo de processamento de UCP de uma entrada de índice durante uma operação de varredura;
- c_o (cpu operator cost): custo de processamento de UCP para realizar uma operação.

O custo CO de um operador O, em um plano de consulta, é então calculado por uma combinação linear de c_s, c_r, c_t, c_i e c_o:

$$CO = n^{\intercal}c = n_s \cdot c_s + n_r \cdot c_r + n_t \cdot c_t + n_i \cdot c_i + n_o \cdot c_o \quad (1)$$

Onde os valores $n = (n_s, n_r, n_t, n_i, n_o)^{\intercal}$ representam o número de páginas acessadas sequencialmente ou aleatoriamente durante a execução do operador O. O custo total estimado de um plano de consulta é simplesmente a soma dos custos dos operadores individuais no plano de consulta.

A precisão do CO depende da precisão dos c's e dos n's. No PostgreSQL por padrão c_s = 1,0, c_r = 4,0, c_t = 0,01, c_t = 0,005 e c_o = 0,0025. Observa-se que essas unidades de custo foram arbitrariamente configuradas pelos projetistas do otimizador do SGBD sem conhecimento do sistema no qual a consulta está sendo executada.

4.8.1 Modelo de custo sensível à assimetria de leitura e escrita

Por décadas o HDD tem sido a solução dominante para armazenamento secundário em SGBDs. Assim como a maioria dos SGBDRs, o modelo de custos do PostgreSQL foi projetado com base nos custos e propriedades dos HDDs, não considerando em seus algoritmos internos os novos custos de E/S dos SSDs, como: assimetria entre leitura/escrita, baixa latência e baixo custo de E/S para acessos aleatórios.

Um dos problemas causados pelo não reconhecimento das diferenças entre os dispositivos de armazenamento é que, mesmo utilizando discos mais rápidos, como um SSD, os planos de consultas gerados tendem a ser iguais (ver seção 5.6), embora tenha sido alterado o tipo de armazenamento.

Bausch et al. [15] apresenta um novo modelo de custos, para o SGBD PostgreSQL – versão 9.0.1, em que propõe a diferenciação entre custos de leitura e escrita pela criação de novos parâmetros configuráveis no SGBD. Além disso, são realizadas adaptações das funções de custos nos principais algoritmos de acesso a dados (*external sort, hash join, sequential scan e index scan*), para que os novos parâmetros sejam reconhecidos.

Nossa proposta utiliza como base o trabalho de Bausch et al. [15] desenvolvido para o PostgreSQL 9.0.1 e implementa essas extensões no PostgreSQL versão 10.1. Em [81] está disponível a versão com as extensões implementadas nesta tese, e a lista de arquivos fontes para os quais foram necessárias alterações está no quadro 4.13. Ressaltamos que, assim como no trabalho de [15], neste trabalho novos parâmetros são inseridos no SGBD, em substituição a outros, visando a diferenciação dos custos de leitura e escrita, bem como a diferenciação de leitura e escrita de forma sequencial e aleatória, conforme listado no quadro 4.14. Também são modificadas as funções de custos dos principais algoritmos internos, como: *external sort, hash join, sequential scan, index scan e parallel index scan* – este último presente somente a partir da versão 10.0 do PostgreSQL.

Quadro 4.13. Lista de fontes do PostgreSQL 10.1 alterados para desenvolvimento das extensões.

Diretório(s) / arquivo.extensão
configure
configure.in
doc/src/sgml/config.sgml
doc/src/sgml/indexam.sgml
doc/src/sgml/perform.sgml
doc/src/sgml/ref/alter_tablespace.sgml
doc/src/sgml/release-8.2.sgml
src/backend/access/common/reloptions.c
src/backend/optimizer/path/costsize.c
src/backend/utils/adt/selfuncs.c
src/backend/utils/cache/spccache.c
src/backend/utils/misc/guc.c
src/backend/utils/misc/postgresql.conf.sample
src/bin/psql/tab-complete.c
src/include/commands/tablespace.h
src/include/optimizer/cost.h
src/include/utils/spccache.h
src/test/regress/input/tablespace.source
src/test/regress/output/tablespace.source

Quadro 4.14. Antigos e novos parâmetros do modelo de custos do SGBD PostgreSQL.

Parâmetros do modelo-padrão	**Parâmetros do novo modelo**
c_s - custo para buscar sequencialmente uma página no disco;	c_{sr} - custo de leitura de uma página em disco de forma sequencial c_{sw} - custo de escrita de uma página em disco de forma sequencial;
c_r - custo para buscar aleatoriamente uma página no disco.	c_{rr} - custo de leitura de uma página em disco de forma aleatória c_{rw} - custo de escrita de uma página em disco de forma aleatória.

4.8.2 Algoritmo de calibração

O novo modelo de custos fornece novos parâmetros de custos configuráveis e a escolha de um conjunto adequado de coeficientes para estes parâmetros é de suma importância. Infelizmente, configurações ótimas dos parâmetros não podem ser calculadas analiticamente porque elas dependem de operações algorítmicas, de propriedades de *hardware* e do sistema operacional, fatores que não são conhecidos. A realização de uma busca exaustiva também é desaconselhável, pois existe um espaço virtualmente infinito de configurações, sem mencionar o longo tempo necessário.

Neste trabalho, em razão do grande espaço de busca associado à procura pelo melhor conjunto de coeficientes relacionados aos novos e antigos parâmetros, foi utilizado um AG.

Algoritmo genético para calibração

De forma geral, para ajustar os parâmetros durante o processo de calibração, o AG gera um conjunto-solução com o objetivo de minimizar o custo total das consultas da carga de trabalho. O AG implementado, a cada geração de indivíduos altera os valores de alguns campos (genes), por meio de operadores genéticos, e posteriormente avalia a nova população de indivíduos (soluções candidatas) até que seja atingido o limite de gerações definido.

Estrutura do indivíduo

Cada indivíduo (cromossomo) neste algoritmo é composto por sete genes e cada um representa parâmetros selecionados de configuração do SGBD. No quadro 4.15 são relacionados todos os genes, a sigla utilizada e sua descrição, sendo que os genes de 1 a 4 se referem a novos parâmetros propostos no trabalho de [15] e não existem na versão padrão do PostgreSQL. Os genes de 5 a 7 são parâmetros referentes ao custo de UCP e já existem na versão padrão do PostgreSQL. No entanto, esses genes foram incluídos no processo de calibração pela sua sensibilidade a alterações nos dispositivos de armazenamento secundário.

Operadores genéticos e avaliação da população

Para a evolução da população de indivíduos, operadores genéticos foram utilizados para seleção e *crossover*. Como método de seleção foi utilizado o torneio estocástico, por ser um dos melhores métodos de seleção encontrado na literatura. O tipo de crossover utilizado

foi de dois pontos. Ademais, foi utilizada uma estratégia de reprodução elitista em que o melhor indivíduo de cada geração é transportado à próxima geração sem nenhuma alteração.

Quadro 4.15. Estrutura e características de cada indivíduo do AG.

Gene	**Sigla**	**Significado**
G1	c_{sr}	custo de leitura de uma página em disco de forma sequencial;
G2	c_{sw}	custo de escrita de uma página em disco de forma sequencial;
G3	c_{rr}	custo de leitura de uma página em disco de forma aleatória;
G4	c_{rw}	custo de escrita de uma página em disco de forma aleatória;
G5	c_{t}	custo de processamento de UCP de uma tupla;
G6	c_{o}	custo para executar um operador;
G7	c_{i}	custo de processamento de UCP de uma entrada de índice em uma operação de varredura de índice.

Após o processo de recombinação, é iniciado o processo de mutação. Nesta etapa, utilizou-se a técnica mutação simples, em que apenas 1 gene é alterado.

Finalizado o processo de mutação, cada indivíduo é avaliado por meio de uma função de avaliação. Esta tem como objetivo pontuar a qualidade *(fitness)* de acordo com a aptidão de cada indivíduo (cromossomo), verificando o percentual de melhoria obtido pelo conjunto de coeficientes presente em cada indivíduo, quando aplicado no SGBD.

Para compor a carga de trabalho utilizada pelo AG, foram escolhidas as consultas mais representativas e que exigem mais esforço computacional entre as consultas do *benchmark* TPC-H, a saber: Q4, Q5, Q9, Q18, Q20 e Q21.

Para calcular o *fitness* de um indivíduo, aplica-se os seus coeficientes no SGBD, e posteriormente é realizada uma avaliação através da extração do custo de cada uma das 10 consultas (utilizando o utilitário *Explain*). Então, para cada consulta é calculado o ganho ou a perda percentual em relação à última melhor solução (indivíduo). Desta forma, o *fitness* do indivíduo é o somatório de ganho ou perda percentual em cada consulta. Em cada geração o indivíduo com maior *fitness* é considerado elite, sendo este transportado para a próxima geração sem alterações. Este processo é realizado até o limite de gerações estabelecido parametricamente e, ao final, o indivíduo com maior *fitness* será dado como solução ótima.

Parâmetros do AG e população inicial

Para a população inicial, foi definido empiricamente um conjunto de 40 indivíduos. A geração da população inicial se deu em 50% de forma aleatória e o restante de forma manual (por especialista). A estratégia de gerar alguns indivíduos de forma manual é uma técnica que visa agilizar o aprendizado, pois utiliza o conhecimento de especialista na área possibilitando melhor convergência e qualidade dos indivíduos da população.

Após a geração da população inicial de classificadores foi necessário que seus valores de energia fossem ajustados. Para tanto, foi criado um conjunto de treinamento com a mesma base de dados, contendo consultas similares às consultas do *benchmark* TPC-H. O processo de ajuste consistiu em executar o AG durante 10 gerações, utilizando o conjunto de treinamento.

Para o conjunto de teste, foram utilizadas 11 consultas do *benchmark* TPC-H, na qual o AG foi executado durante 30 gerações e a percentagem de *crossover* foi de 80% com mutação de 2%. Na tabela 4.2 são apresentados os valores ótimos obtidos pelo AG, os quais foram avaliados e utilizados em um dos cenários de teste propostos no capítulo 5 seção 5.2.3.

Tabela 4.2. Configurações ótimas obtidas durante o processo de calibração.

Parâmetros-padrão SGBD	**Parâmetros novo modelo**
c_s = 1.0000	c_{sr} = 0.79000
	c_{sw} = 25.90000
c_r = 4.0000	c_{rr} = 4.60000
	c_{rw} = 10.32000
c_t = 0,01	c_t = 0,00003
c_o = 0,0025	c_o = 0,00008
c_i = 0,005	c_i = 0,00420

4.9 Considerações finais do capítulo

Neste capítulo foi apresentada a abordagem metodológica empregada na realização do *tuning* de índices em ambientes híbridos de armazenamento, apresentando as principais etapas do processo, os componentes principais do algoritmo ITLCS proposto, sua arquitetura e a integração entre todos os componentes.

Além dos módulos básicos de um SC baseado em Holand [47], neste trabalho também foram desenvolvidos alguns módulos adicionais como: repositório base, módulo de heurísticas e um módulo de monitoramento da carga de trabalho. Esses módulos adicionais têm por objetivo adaptar o SC ao problema, possibilitando assim que o algoritmo ITLCS

implementado, realize o *tuning* de índices em SGBDRs que utilizam armazenamento secundário híbrido (HDD e SSD).

No módulo de descoberta de novas regras, o AG desenvolvido utiliza mecanismos inspirados na natureza como reprodução e mutação, para obter a evolução da população de classificadores/regras atuais, descobrindo novas regras cada vez mais ajustadas aos ambientes, possibilitando que o *tuning* de índices seja especializado e direcionado aos ambientes híbridos de armazenamento. Desta forma, espera-se que a população de classificadores direcionados a HDD apresente características diferentes daquelas direcionadas a SSD, evidenciando a adaptabilidade do SC aos ambientes híbridos de armazenamento, como proposto inicialmente.

Alterações realizadas no SGBD PostgreSQL foram detalhadas, evidenciando as principais características do modelo de custo proposto em [15], que foi adaptado e utilizado neste trabalho. Foi também apresentado o processo de calibração implementado, usando um AG, que possibilitou encontrar valores úteis para os coeficientes, sobretudo dos novos parâmetros do modelo de custos implementado, os quais são utilizados no próximo capítulo, concernente à avaliação experimental.

Capítulo 5

Resultados Experimentais

Este capítulo apresenta o método utilizado para avaliação experimental do algoritmo ITLCS, proposto neste trabalho. Nas seções subsequentes serão detalhados os seguintes itens: as configurações do ambiente experimental; a carga de trabalho utilizada; as ferramentas empregadas como base de comparação; os cenários de testes; os resultados obtidos e a discussão detalhada dos achados, a partir da qual discute-se a validade das hipóteses de pesquisa.

5.1. Ambiente experimental

Os experimentos foram realizados utilizando um computador servidor Hewlett Packard (2,5 GHz Intel Xeon Quad-Core, 8Gb de RAM), funcionando em um sistema operacional GNU/Linux kernel 2.6 (64 bits) e SGBD PostgreSQL versão 10.1 com a extensão Hypopg [42] instalada. O algoritmo ITLCS foi desenvolvido utilizando a plataforma Java 2 Standard Edition version 8.

Como este trabalho é direcionado a ambientes híbridos de armazenamento, o servidor possui quatro discos, sendo dois HDDs (1 TB 6 GB/s Sata III) e dois SSDs (500 GB 6 GB/s Sata III). O sistema operacional e o SGBD foram instalados no SSD.

Adotou-se como carga de trabalho o OSDL DBT3 [33], uma implementação *open-source* do TPC-H *benchmark* [82], ajustada para o SGBD PostgreSQL, com o fator de escala

40 (40 GB). O TPC-H *benchmark* é comumente utilizado para avaliar sistemas de banco de dados, pois provê um conjunto de 22 consultas de perfil analíticas, comuns em ambientes de *Data Warehouse* e sistemas de apoio a decisão, detalhadas no Anexo A.

Para definir o fator de escala, foram realizados experimentos preliminares com as escalas 10, 20, 40 e 80, e baseado nos experimentos foi escolhido o fator 40 – ocupa aproximadamente 40 GB de armazenamento, que trouxe resultados significativos na utilização de índices, além de não ocupar tanto espaço de armazenamento como utilizando a escala 80.

Nos experimentos realizados utilizando o TPC-H, para calcular o tempo médio de cada consulta, executamos a mesma consulta sete vezes e calculamos a média das execuções excluindo as duas primeiras execuções, que não são consideradas devido ao processo de *"warm up"* em que os dados são acomodados em cache. Após, os *caches* do SGBD e do sistema operacional são apagados, para então ser executada uma nova consulta SQL, para evitar a interferência entre uma consulta SQL e outra.

5.2. ITLCS versus configuração somente com índices primários

Esta subseção tem por objetivo avaliar os benefícios trazidos pelos índices recomendados pelo ITLCS e pelas regras geradas por ele, tanto em um ambiente de armazenamento HDD quanto de SSD. Além disso, avalia-se o ITLCS utilizando o SGBD PostgreSQL versão padrão e versão estendida (capítulo 4).

5.2.1. Ambiente usando HDD e SGBD PostgreSQL padrão

Para avaliar o desempenho do ITLCS em ambiente com HDD, foram estabelecidos dois cenários, a considerar a execução da carga de trabalho em uma base de dados, contendo:

a) somente os índices primários – resultantes das chaves primárias definidas no modelo relacional do *benchmark* TPC-H;
b) os índices primários do *benchmark* e índices secundários recomendados pelo ITLCS.

Na figura 5.1 é apresentado o somatório do tempo médio de execução de todas as consultas da carga de trabalho para os dois cenários, respectivamente. É possível verificar que utilizando HDD e PostgreSQL padrão, o ITLCS não foi capaz de reduzir o tempo total, comparado ao cenário utilizando somente índices primários.

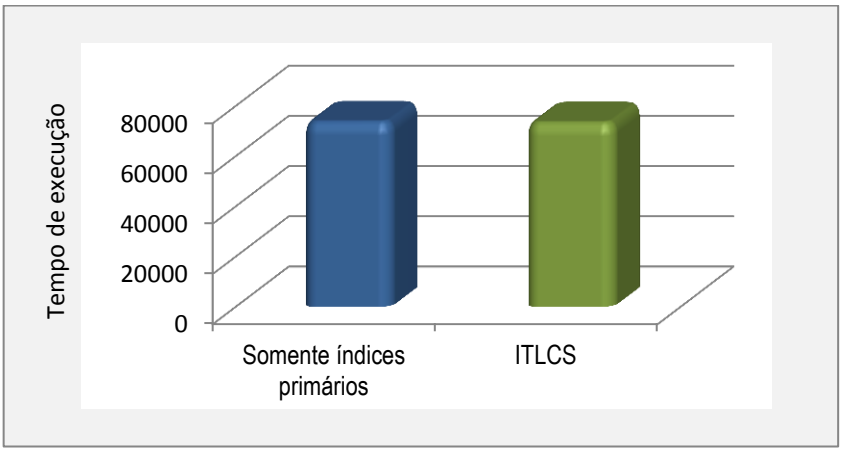


Figura 5.1. Desempenho geral do ITLCS em HDD.
Fonte: o autor.

Na figura 5.2 é apresentado o tempo individual de cada consulta, mas apenas para as consultas que tiveram índices recomendados pelo ITLCS.

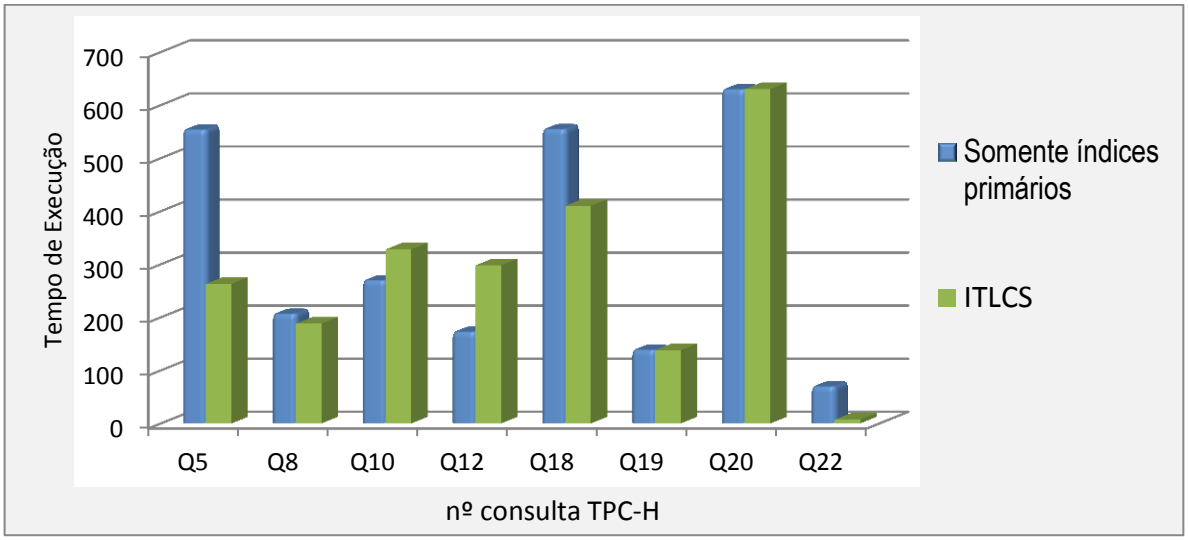


Figura 5.2. Desempenho individual das consultas do TPC-H em HDD.

Fonte: o autor.

Como detalhado na tabela 5.1, a utilização dos índices criados pelo ITLCS, contribuíram para reduzir o tempo de execução das consultas Q5, Q8, Q18 e Q22 em 52%, 8%, 25% e 90%, respectivamente, se comparados ao cenário que utiliza somente índices primários. No entanto, para as consultas Q10 e Q12, a utilização dos índices causaram baixa no desempenho, reduzindo o ganho geral (Figura 5.1) obtido.

5.2.2. Ambiente usando SSD e SGBD PostgreSQL padrão

Para avaliar a contribuição trazida pelo ITLCS para ambiente com SSD usando PosgreSQL padrão, foram estabelecidos dois cenários, a considerar a execução da carga de trabalho em uma base de dados, contendo:

a) somente índices primários do *benchmark*;
b) os índices primários do *benchmark* e índices secundários recomendados pelo ITLCS.

Na figura 5.3 é apresentado o somatório do tempo médio de execução de todas as consultas da carga de trabalho para estes dois cenários. Diferentemente do ambiente com HDD, no SSD o ITLCS obteve significativos 20% de ganho de desempenho, ou seja, o tempo total de execução da carga de trabalho foi reduzido em 20% em relação à configuração que utiliza somente índices primários.

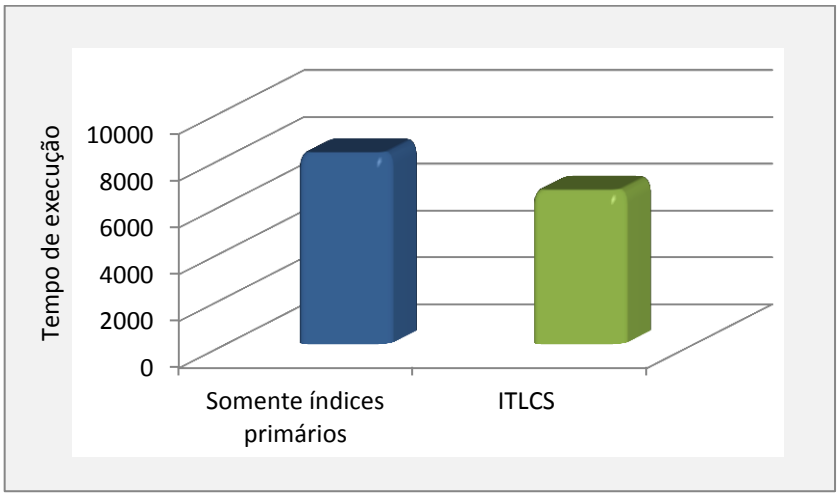


Figura 5.3. Desempenho geral do ITLCS em SSD.

Fonte: o autor.

É apresentado na figura 5.4 o tempo individual de cada consulta, mas apenas para as consultas que tiveram índices recomendados pelo ITLCS.

Como detalhado na tabela 5.1 e apresentado graficamente na figura 5.4, destacamos as consultas Q3, Q5, Q8, Q18, Q21 e Q22, que obtiveram melhoria significativa: de 28%, 84%, 34%, 59%, 19% e 89%, respectivamente, se comparadas ao cenário que utiliza somente índices primários. Diferentemente do ambiente com HDD, utilizando SSD, o ITLCS recomendou índices para a maioria das consultas. Além disso, o conjunto de índices encontrados, com exceção das consultas Q12 e Q20, trouxeram incremento no desempenho.

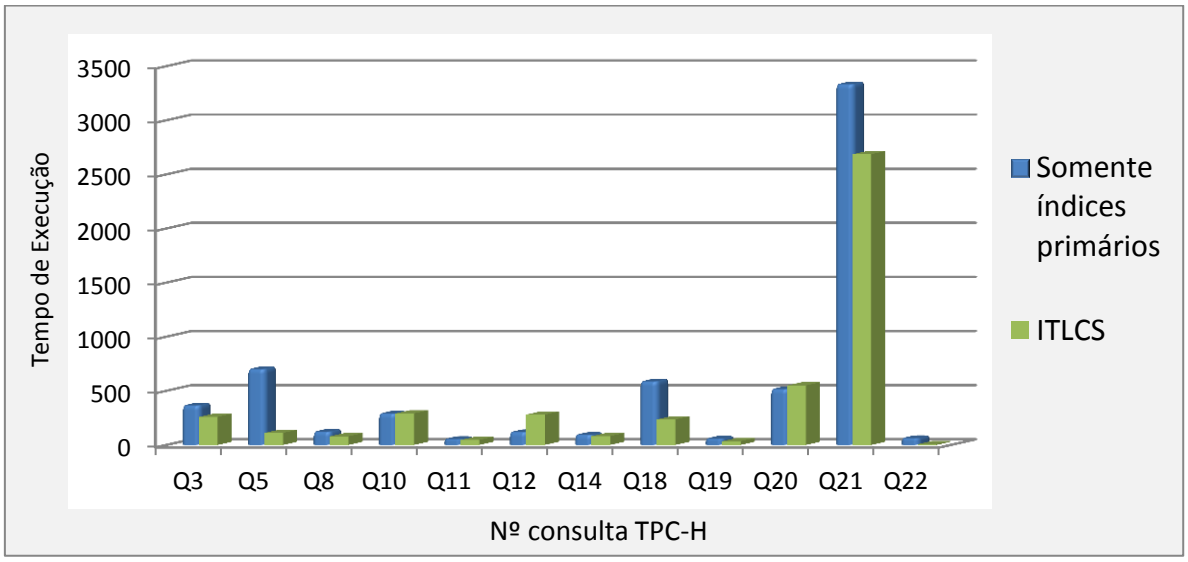


Figura 5.4. Desempenho individual das consultas do TPC-H em SSD.
Fonte: o autor.

5.2.3. Ambiente usando SSD e SGBD PostgreSQL estendido

Nesta subseção pretende-se avaliar se as alterações no PostgreSQL – propostas no capítulo 5 – trouxeram benefícios de redução do tempo de execução da carga de trabalho. Para isso, foram estabelecidos quatro cenários. Os dois primeiros (itens a e b) são os mesmos da subseção anterior e foram utilizados somente para servir como base de comparação. Assim, considera-se a execução da carga de trabalho em uma base de dados, contendo:

a) somente índices primários do *benchmark* e utilizando o PostgreSQL padrão;
b) os índices primários do *benchmark*, e índices secundários recomendados pelo ITLCS, utilizando o PostgreSQL padrão;
c) somente índices primários do *benchmark* e utilizando o PostgreSQL estendido;
d) os índices primários do *benchmark*, e índices secundários recomendados pelo ITLCS, utilizando o PostgreSQL estendido.

Apresenta-se na figura 5.5 o somatório da média de tempo de execução de todas as consultas da carga de trabalho para os cenários estabelecidos. É possível observar que o ITLCS utilizando o PostgreSQL estendido, obteve o melhor resultado, se comparado aos outros cenários.

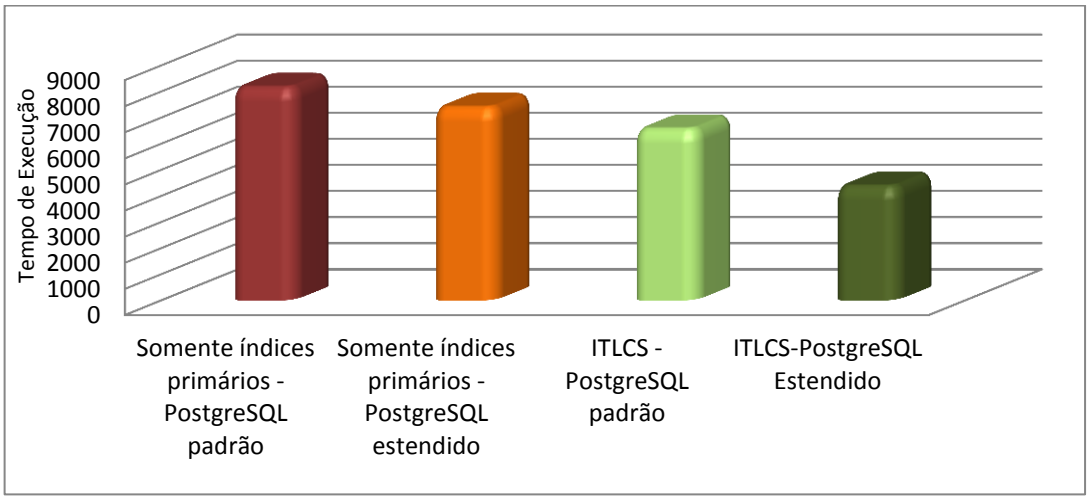


Figura 5.5. Comparativo de desempenho entre o SGBD PostgreSQL padrão e estendido.
Fonte: o autor.

Para ser possível avaliar o benefício proporcionados pelo PostgreSQL estendido em cada consulta individual, na figura 5.6 é apresentada uma comparação do tempo de execução usando o PostgreSQL padrão e o estendido, bem como os respectivos índices recomendados pelo ITLCS.

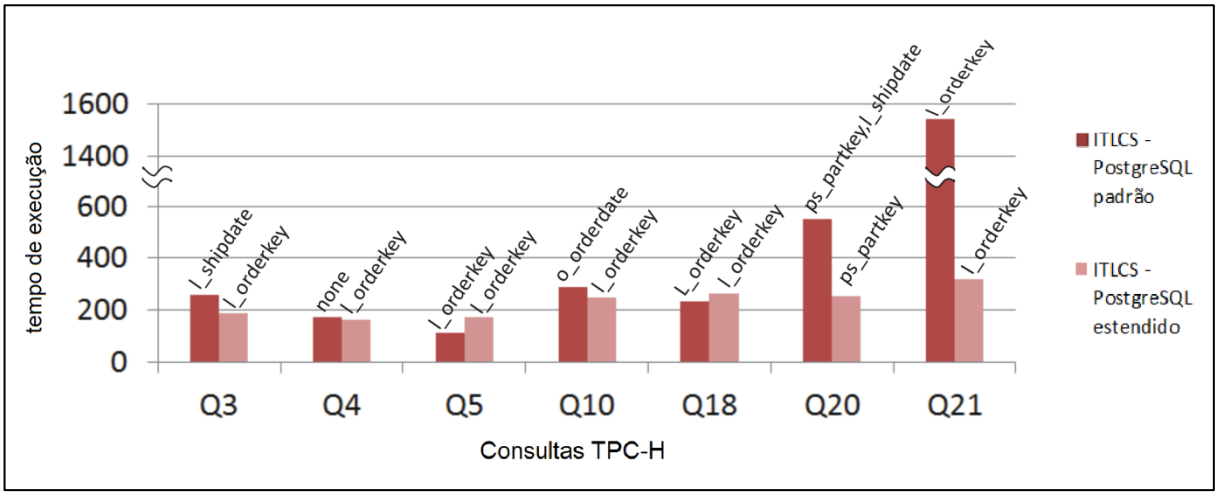


Figura 5.6. Comparativo por consulta, usando o PostgreSQL padrão e estendido.
Fonte: o autor.

5.2.4. Desempenho do ITLCS em treinamento e teste

Embora abordagens utilizando algoritmos evolutivos tenham como desvantagem o tempo de convergência na obtenção da solução, neste trabalho isso foi significativamente reduzido. Graças ao emprego de índices hipotéticos e ao armazenamento de metadados de

informações utilizadas com mais frequência, reduziu-se a extração de informações redundantes (já pesquisadas) utilizadas pelo subsistema de descoberta de novas regras.

Na figura 5.7 é apresentado o desempenho do ITLCS durante as fases de treinamento e teste, para os ambientes que utilizam HDD e SSD. Um ponto de destaque observado na figura 5.7 é que somente durante a primeira iteração da fase de treinamento foi necessário um tempo significativo, em virtude da coleta de estatísticas e persistência dos metadados (citados no parágrafo anterior) naquela etapa. Na fase de treinamento todas as regras partem com um mesmo valor de energia. Através da execução de dez ciclos de iterações e do processo de descoberta de novas regras utilizando o AG, os valores de energia dos classificadores são ajustados. Com as regras já ajustadas, inicia-se a fase de teste do ITLCS.

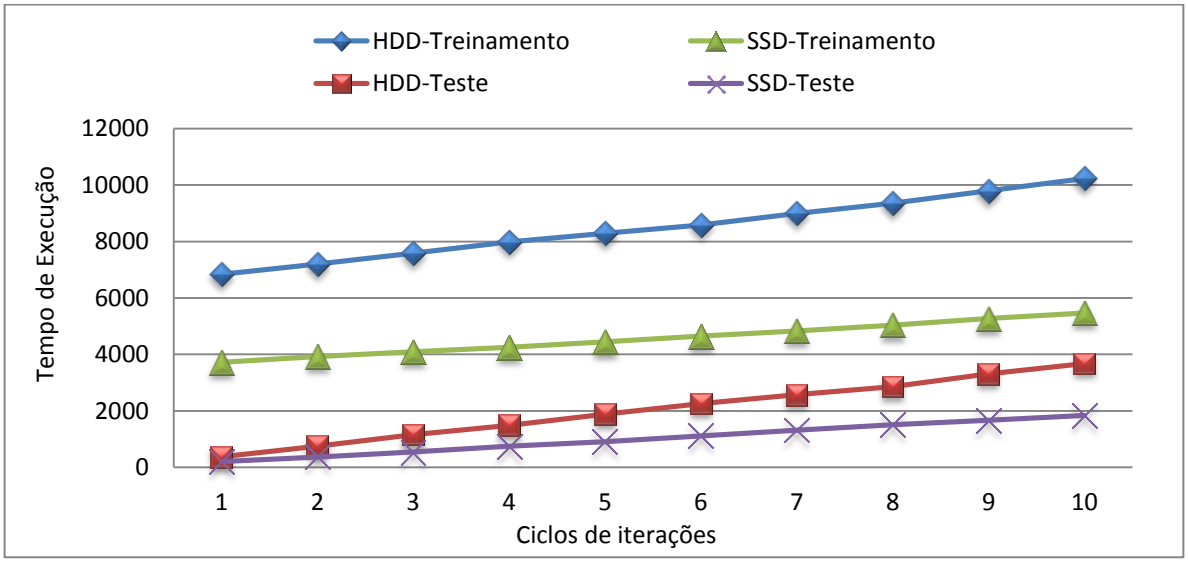


Figura 5.7. Desempenho do ITLCS nas fases de treinamento e teste.

Fonte: o autor.

5.3. ITLCS versus EDB versus POWA

Visto que não foram encontradas na literatura, soluções que utilizam Sistemas Classificadores aplicados a *tuning* de índices, optou-se por avaliar o ITLCS, comparando-o com outras soluções, que também realizam o *tuning* de índices, e que são compativeis para uso no SGBD PostgreSQL, utilizado neste trabalho. Entre as ferramentas analisadas, foram escolhidas as ferramentas EnterpriseDB Index Advisor (EDB) [16] e PostgreSQL Workload Analyzer (POWA) [17], que sao soluções amplamente utilizadas por DBAs do SGBD PostgreSQL.

5.3.1. Ambiente usando HDD e PostgreSQL padrão

Para avaliar o desempenho do ITLCS em comparação com as ferramentas EDB e POWA utilizando o PostgreSQL padrão, foram estabelecidos quatro cenários, a considerar a execução da carga de trabalho em uma base de dados, contendo:

a) somente os índices primários do *benchmark*;
b) os índices primários do *benchmark* e índices secundários recomendados pelo ITLCS;
c) os índices primários do *benchmark*, e índices secundários recomendados pela ferramenta EDB;
d) os índices primários do *benchmark*, e índices secundários recomendados pela ferramenta POWA.

Na figura 5.8 é apresentado o somatório do tempo médio de execução de todas as consultas da carga de trabalho para os cenários estabelecidos. Conforme pode ser visualizado, considerando o tempo total de execução de todas as consultas, nenhuma das soluções avaliadas (EDB, POWA e ITLCS) obteve vantagens em suas recomendações de índices, quando utilizando armazenamento HDD.

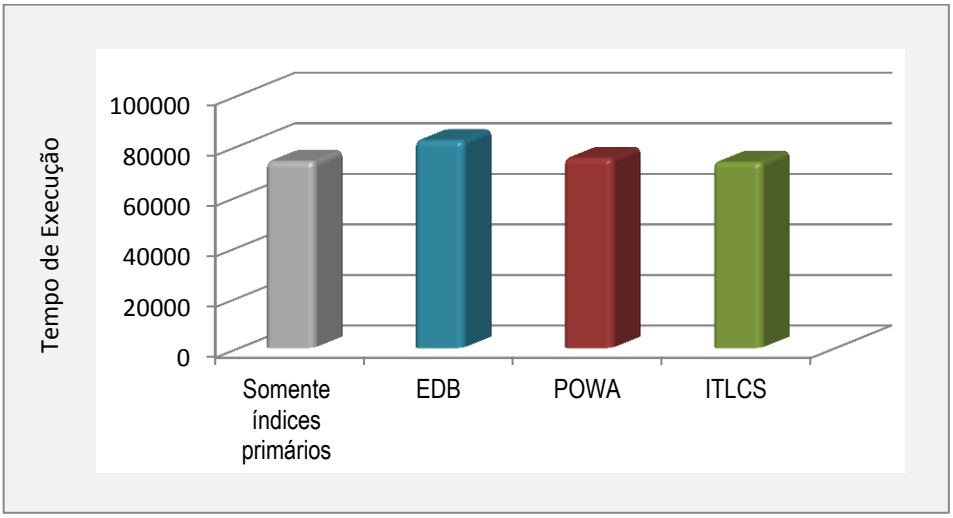


Figura 5.8. Comparativo de desempenho do ITLCS com outros métodos em HDD.
Fonte: o autor.

Para melhor visualizar os resultados individuais de cada consulta, no ambiente utilizando HDD, as figuras 5.9 e 5.10 apresentam os resultados divididos em dois grupos, de acordo com o tempo necessário para execução.

Como detalhado na tabela 5.1 e apresentado nas figuras 5.9 e 5.10, os índices criados pelo ITLCS tiveram desempenho superior às ferramentas EDB e POWA para a maioria das

consultas. No entanto, para as consultas Q10 e Q12 o desempenho piorou, reduzindo assim, uma parte do ganho geral obtido.

O fato de alguns índices criados ocasionarem piora no desempenho também foi verificado nas ferramentas POWA e EDB, destacando as consultas Q4, Q8 e Q19, nas quais os índices criados causaram significativa piora no desempenho.

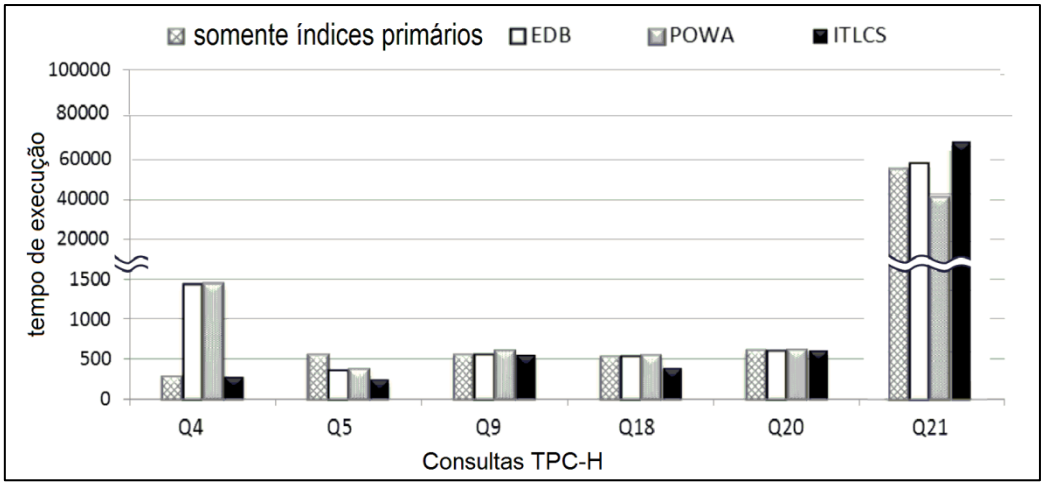


Figura 5.9. Desempenho por consulta, usando HDD – grupo 1.
Fonte: o autor.

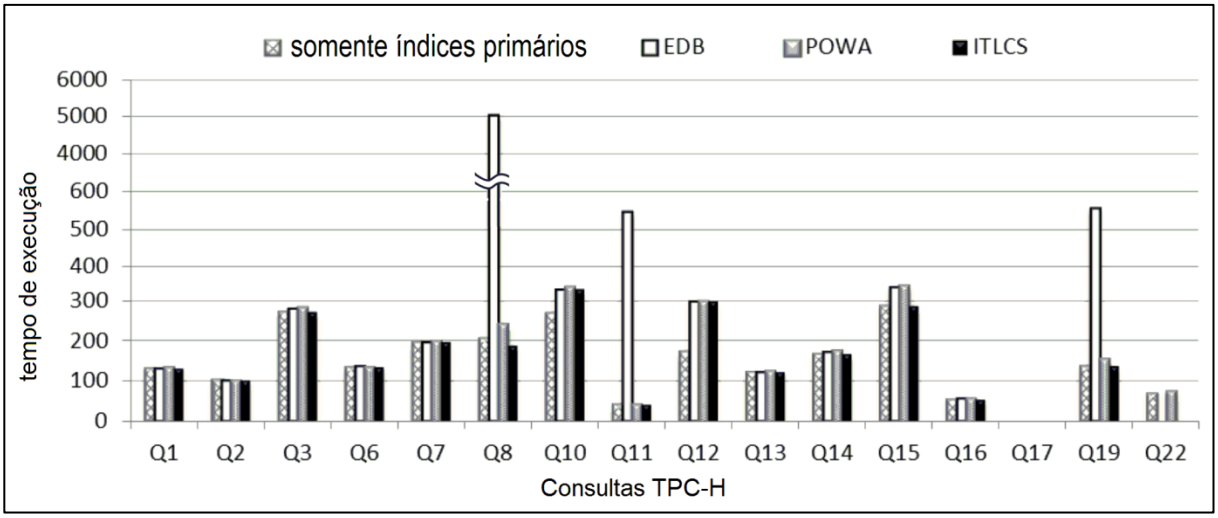


Figura 5.10. Desempenho por consulta, usando HDD – grupo 2.
Fonte: o autor.

5.3.2. Ambiente usando SSD e PostgreSQL padrão

Assim como na sessão anterior, para avaliar o ITLCS em comparação com a ferramenta EDB e POWA, utilizando em ambas o SSD, foram estabelecidos quatro cenários, a considerar a execução da carga de trabalho em uma base de dados, contendo:

a) somente os índices primários do *benchmark*;
b) somente os índices primários do *benchmark* e índices secundários recomendados pelo ITLCS;
c) os índices primários do *benchmark*, e índices secundários recomendados pela ferramenta EDB;
d) os índices primários do *benchmark* e índices secundários recomendados pela ferramenta POWA.

Na figura 5.11 é apresentado o somatório do tempo médio de execução de todas as consultas da carga de trabalho para os cenários estabelecidos.

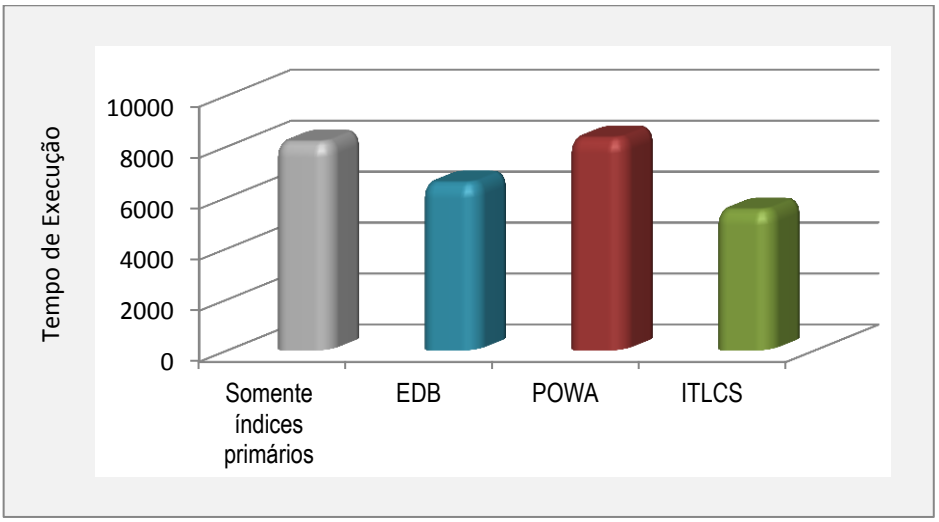


Figura 5.11 – Comparativo de desempenho do ITLCS com outros métodos em SSD.
Fonte: o autor.

Conforme apresentado na figura 5.11, usando ambiente com SSD, e considerando o tempo total de execução de todas as consultas, o ITLCS obteve significativa redução de 20% se comparado ao primeiro cenário – que usa somente índices primários e também obteve melhores resultados que as ferramentas EDB e POWA.

As figuras 5.12 e 5.13 apresentam os resultados individuais. As consultas foram separadas em dois grupos, assim como na subseção anterior. Os resultados mostram que a ferramenta EDB melhorou o desempenho de nove consultas. No entanto, para outras nove consultas as recomendações causaram piora no desempenho. Por sua vez, a ferramenta POWA conseguiu melhorar o desempenho de sete consultas, mas para oito consultas o desempenho piorou. Uma nítida desvantagem da ferramenta POWA é o número de índices

recomendados e não utilizados pelo SGBD, totalizando 15 ocorrências, como detalhado na tabela 5.1, destacados com um asterisco.

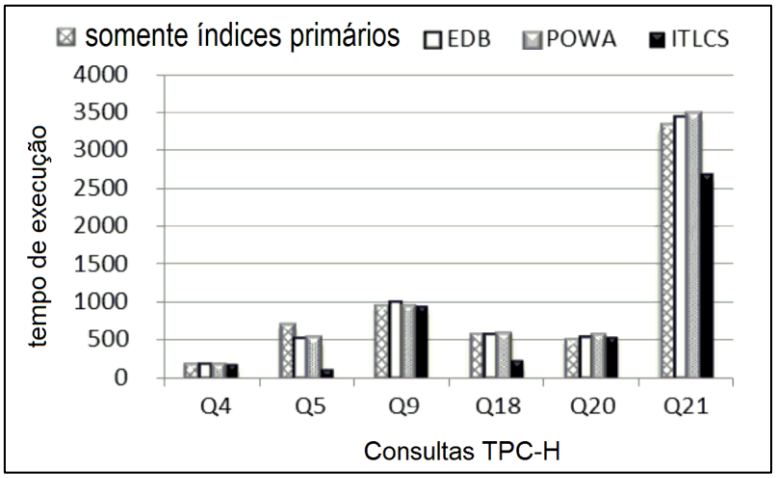


Figura 5.12. Desempenho por consulta, usando armazenamento em SSD – grupo 1.
Fonte: o autor.

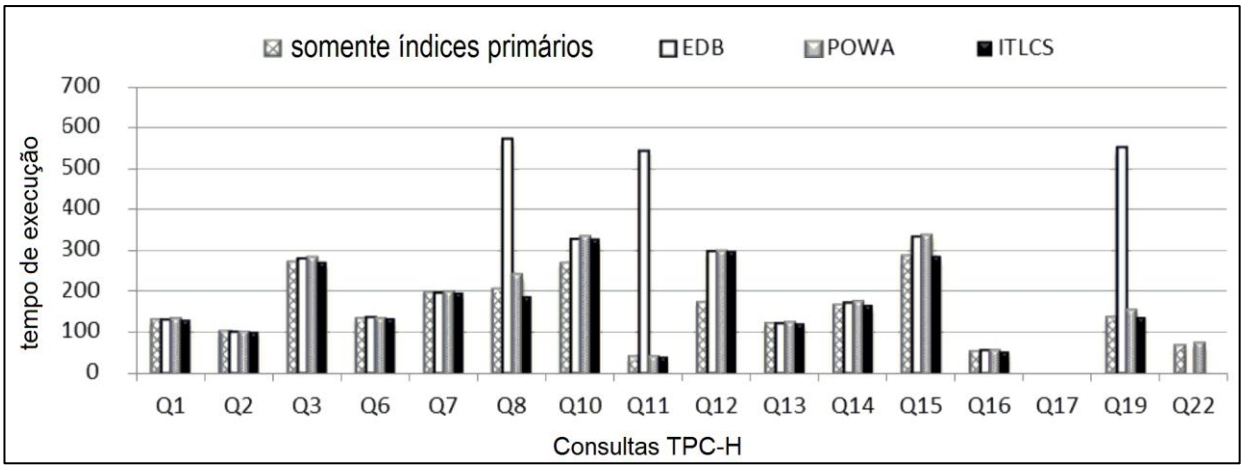


Figura 5.13. Desempenho por consulta, usando armazenamento em SSD – grupo 2.
Fonte: o autor.

5.4. ITLCS executando migração de índices

Para avaliar ações de migração de índices de um dispositivo A para um dispositivo B, mais ágil, foi utilizado o cenário que conta com armazenamento de tabelas e índices em HDD e permite que índices sejam migrados para SSD.

Como apresentado na figura 5.14, após a execução do *benchmark* o ITLCS recomendou a migração de índices para duas consultas e em ambas foi sobre índices da maior tabela do *benchmark* – tabela *lineitem* com 37 GB – que consequentemente tem os maiores índices, sendo desta forma bons candidatos para um processo de migração. Para a consulta Q5, foi migrado o índice da coluna *l_orderkey*, de tamanho 5GB, equivalente a 14% da tabela.

Tabela 5.1. Conjunto de índices recomendados por ITLCS, EDB e POWA.

		EDB		POWA		ITLCS	
QN	**Tipo Disco**	**Tempo %**	**Coluna indexada**	**Tempo%**	**Coluna indexada**	**Tempo %**	**Coluna indexada**
Q1	HDD	-	Nenhum	-	l_shipdate*	-	Nenhum
	SSD	-	Nenhum	-	l_shipdate*	-	Nenhum
Q2	HDD	-	Nenhum	-	p_size*	-	Nenhum
	SSD	55%	(p_parkey,p_size)	-	P_size*, R_name*	-	Nenhum
Q3	HDD	-2%	c_mktsegment	-2%	c_mktsegment	-	Nenhum
	SSD	-6%	c_mktsegment, o_orderdate *	17%	l_shipdate, o_orderdate*	28%	l_shipdate
Q4	HDD	-372%	o_orderdate	-372%	o_orderdate	-	Nenhum
	SSD	4%	o_orderdate	7%	o_orderdate, l_orderkey	-	Nenhum
Q5	HDD	32%	o_orderdate	32%	o_orderdate	52%	L_orderkey
	SSD	23%	o_orderdate	23%	o_orderdate. o_orderkey	84%	L_orderkey
Q6	HDD	1%	(l_shipdate, quantity,l_discount	-2%	(l_shipdate,l_discount)	-	Nenhum
	SSD	-31%	(l_shipdate,l_quantity,l_discount)	-33%	(l_shipdate,l_discount,l_quantity)	-	Nenhum
Q7	HDD		Nenhum		L_shipdate*	-	Nenhum
	SSD	23%	L_shipdate	23%	L_shipdate	-	Nenhum
Q8	HDD	-2333%	o_orderdate, l_partkety	-16%	p_type, o_orderdate	8%	P_type
	SSD	24%	p_type, l_partkey	33%	p_type, (o_orderkey*, o_orderdate)*	34%	P_type
Q9	HDD	-	Nenhum	-6%	p_name*	-	Nenhum
	SSD	-5%	l_partkey*	0%	p_name*	-	Nenhum
Q10	HDD	-21%	o_orderdate	-23%	l_returnflag *, o_orderdate	-21%	O_orderdate
	SSD	-19%	O_orderdate, l_returnflag *	-19%	l_returnflag *	0%	O_orderdate
Q11	HDD	-1221%	Ps_suppkey	-	Nenhum	-	Nenhum
	SSD	16%	Ps_suppkey	-	Nenhum	16%	ps_suppkey
Q12	HDD	-73%	L_receiptdate	-72%	(L_receiptdate,l_shipmode)	-73%	l_receiptdate
	SSD	-153%	L_receiptdate	-141%	(L_receiptdate,l_shipmode)	-137%	L_receiptdate, l_shipmode
Q13	HDD	-	Nenhum	-	O_comment*	-	Nenhum
	SSD	-	Nenhum	-	O_comment*	-	Nenhum
Q14	HDD	-4%	L_shipdate	-4%	L_shipdate	-	Nenhum
	SSD	13%	L_shipdate	13%	L_shipdate	13%	L_shipdate
Q15	HDD	-17%	L_shipdate	-17%	L_shipdate	-	Nenhum
	SSD	-116%	L_shipdate	-116%	L_shipdate	-	Nenhum
Q16	HDD	-2%	P_size	-2%	P_size, P_type*	-	Nenhum
	SSD	-11%	P_size	-16%	P_size, P_brand*, P_type*, S_comment*	-	Nenhum
Q17	HDD		P_brand, L_partkey	0%	(p_brand, p_containner)	-	Nenhum
	SSD	0m0s	P_brand, L_partkey	0%	P_brand	-	Nenhum
	SSD	0m0s	P_brand, L_partkey	0%	P_brand	-	Nenhum
Q18	HDD	0%	Nenhum	-	Nenhum	25%	L_orderkey
	SSD	0%	Nenhum	-	Nenhum	59%	L_orderkey
Q19	HDD	-1342%	(p_containner,p_brarand,p_size), l_partkey	-11%	(l_shipinstruct, l_shipmode) (p_brand, p_container,p_size) l_quantity*, p_size*	0%	p_size,l_partkey, p_brand, p_container, L_shipmode* l_shipinstruct*
	SSD	98%	(container,brand,size), l_partkey	-71%	(l_shipinstruct, l_shipmode), (p_size,p_brand,p_container)*, L_quantity*	42%	p_brand, p_size, p_container, l_partkey, l_shipinstruct*
Q20	HDD	1%	s_name, l_shipdate	0%	l_shipdate, p_name *	0%	ps_partkey, l_shipdate
	SSD	-6%	s_name, l_shipdate	-11%	l_shipdate, p_name *	-6%	ps_partkey, l_shipdate
Q21	HDD		S_nationkey		L_orderkey, o_orderstatus	-	Nenhum
	SSD	-3%	L_suppkey*	-4%	L_orderkey,(o_orderkey, o_orderstatus)*, l_suppkey*	19%	L_orderkey
Q22	HDD	90%	O_custkey,c_acctbal	-4%	C_acctbal	90%	O_custkey, c_acctbal
	SSD	89%	O_custkey,c_acctbal	43%	C_acctbal	89%	O_custkey, c_acctbal

* Índice recomendado, mas não utilizado pelo SGBD ao executar a consulta.

É, portanto, um índice de tamanho considerável e um bom candidato à migração. Na consulta Q18 também foi migrado o mesmo índice – *l_orderkey*.

Após a migração dos índices, constatou-se benefícios com a redução do tempo de execução de 9% e 10%, respectivamente, para as consultas Q5 e Q18.

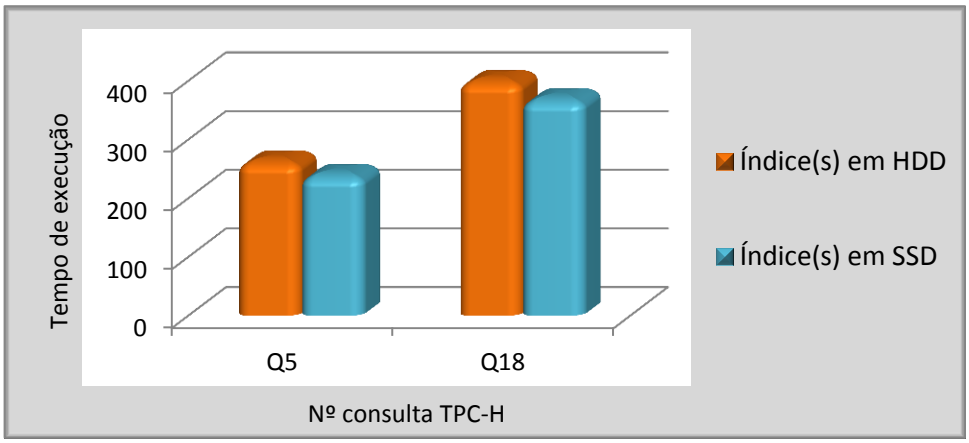


Figura 5.14. Comparativo de desempenho na migração de índices de HDD para SSD.
Fonte: o autor.

5.5. Análise das regras geradas pelo ITLCS

Uma vez que o ITLCS é um algoritmo baseado em regras, uma análise importante a se fazer é a de seu conjunto final de regras (classificadores).

No quadro 5.1 a seguir, para cada ação presente no consequente das regras é apresentada a regra com maior energia encontrada pelo SC. Uma das hipóteses investigadas nesta tese é a de que a migração de um banco de dados de HDD para SSD requer modificação nas regras de *tuning* para que o desempenho seja maximizado. Para comprovar essa hipótese fizemos experimentos utilizando os dois cenários a seguir, usando armazenamento SSD e SGBD PostgreSQL padrão.

- Executamos o SC contendo regras geradas e evoluídas para HDD. Após a execução do SC e da criação dos índices recomendados, foi realizado a execução completa da carga de trabalho, a fim de verificar o tempo total gasto utilizando os índices recomendados.
- Executamos o SC contendo regras geradas e evoluídas para SSD. Após a execução do SC e da criação dos índices recomendados, foi realizado a execução completa da carga de trabalho, para assim medir o tempo total gasto utilizando os índices recomendados.

Quadro 5.1. Melhores regras descobertas

SE (armazenamento) = (HDD) (existe algum índice secundário na coluna) = (não) (qual o tipo de dado da coluna) = (numérico) (qual o grau de seletividade da coluna) = (91% a 100%) (a coluna é utilizada em predicado de junção) = (sim) (qual a percentagem de tuplas retornadas) < (5%) ENTÃO (criar um índice b-tree, utilizando HDD)
SE (armazenamento) = (SSD) (existe algum índice secundário na coluna) = (não) (coluna está em cláusulas de group by) = (sim) (tipo de dado da coluna) = (numérico) (qual o grau de seletividade da coluna) = (86% - 90%) ENTÃO (criar um índice b-tree utilizando SSD)
SE (existe algum índice secundário na coluna) = (sim) (se existe algum índice na coluna, qual seu grau de fragmentação?) = (50-70%) ENTÃO (reconstruir o índice)
SE (armazenamento) = (HDD) (existe algum índice secundário na coluna) = (sim) (se existe algum índice, qual a altura da árvore b-tree?) = (4-5) ENTÃO (migrar índice existente de um HDD para um SSD)

Na tabela 5.2 verifica-se que no cenário 1 – ambiente usando SSD com regras geradas para HDD – a execução da carga de trabalho tomou 2h 2m 34s. No cenário 2 a execução foi mais ágil, demorando 1h 9m 56s. Portanto, no cenário 2 que também utiliza SSD, mas contendo regras adaptadas ao ambiente SSD, o desempenho foi maximizado. Outros dois fatores que confirmam a hipótese são o número de índices gerados e o número de consultas que utilizaram índices. Houve diferenças entre os dois cenários, sendo que a utilização de índices no segundo cenário foi superior à do primeiro.

Tabela 5.2. Avaliação comparativa entre regras geradas para HDD e regras para SSD.

Cenários	**Tempo médio Total**	**Nº de índices**	**Consultas que utilizaram índices**
1. Ambiente usando SSD com regras geradas para HDD	2h 2m 34s	14	8
2. Ambiente usando SSD com regras geradas para SSD	1h 9m 56s	15	12

Outra análise realizada no conjunto final de regras teve o intuito de identificar suas diferenças internas. Nas tabelas 5.3 e 5.4 são sumarizados dados sobre as regras que efetivamente foram aplicadas pelo ITLCS em alguns cenários de testes propostos, para a ação de criar índices, que concentra a maior quantidade de regras do SC. Além disso, são analisados os genes 10 e 12, referentes à quantidade de tuplas retornadas e seletividade, respectivamente. Conforme observado em nossos experimentos empíricos, essas

características foram as que mais apresentaram diferenças entre as regras direcionadas a HDD e SSD. Ademais, segundo os trabalhos de [14], [35] e [45], essas características são as mais impactadas pelas diferenças de custos de E/S, entre dispositivos HDD e SSD.

Tabela 5.3. Sumarização da característica de quantidade de tuplas retornadas nas regras finais.

Quantidade de tuplas retornadas								
Cenários	Quantidade de regras por faixa							
	0 e 5%	6 e 10%	11 e 15%	16 e 20%	21 e 30%	31 e 40%	41 e 60%	61 e 100%
HDD/PostgreSQL padrão	5	0	0	0	0	0	0	0
SSD/PostgreSQL padrão	4	2	0	0	0	0	0	0
SSD/PostgreSQL estendido	3	3	1	0	0	0	0	0

Tabela 5.4. Sumarização da característica de seletividade nas regras finais.

Seletividade								
Cenários	Quantidade de regras por faixa							
	0 e 30%	31 e 60%	61 e 70%	71 e 75%	76 e 80%	81 e 85%	86 e 90%	91 e 100%
HDD/PostgreSQL padrão	0	0	0	0	0	0	0	7
SSD/PostgreSQL padrão	0	0	0	0	0	1	1	8
SSD/PostgreSQL estendido	0	0	0	0	0	1	2	7

A tabela 5.3 apresenta o critério relativo à quantidade de tuplas retornadas. Observou-se que, no cenário com HDD, o limite ficou em 5%. Contudo, para os dois ambientes que utilizam SSDs as regras apresentaram diferenças, totalizando cinco regras na faixa de 6% a 10% e uma regra na faixa de 11% a 15%. De fato, regras encontradas na literatura que consideram somente HDDs indicam que, quando o predicado de seleção afeta até 10% dos registros, o uso de índice se torna aconselhável. Todavia, em uma regra encontrada pelo SC no cenário usando SSD o percentual atingiu 15%, considerado bem acima das regras para HDD. Assim como demonstrado em Lee et al. [14], nos experimentos com SSDs desta tese o uso de índices foi vantajoso, mesmo quando são recuperadas mais de 5% das tuplas de uma tabela.

No critério de seletividade de uma coluna (tabela 5.4) é possível verificar que nas regras para HDD a seletividade variou apenas na faixa de 91 a 100%. Contudo, nas regras para SSD a seletividade atingiu taxas menores, variando de 81% a 85% em duas ocasiões, e de 86% a 90% em três ocasiões. Como esperado, mesmo com uma seletividade menor o uso

de índices em SSD foi vantajoso, ou seja, mesmo que determinada coluna possua de 81 a 90% dos valores distintos o acesso indexado será melhor do que, por exemplo, com o método *Sequential Scan*, que acessa a tabela por completo. Utilizando armazenamento em SSD, devido à ausência de latência mecânica e operações ágeis de leitura, mesmo com uma quantidade menor de valores distintos, o acesso indexado se mostra como melhor opção, evidenciando a relevância das regras encontradas.

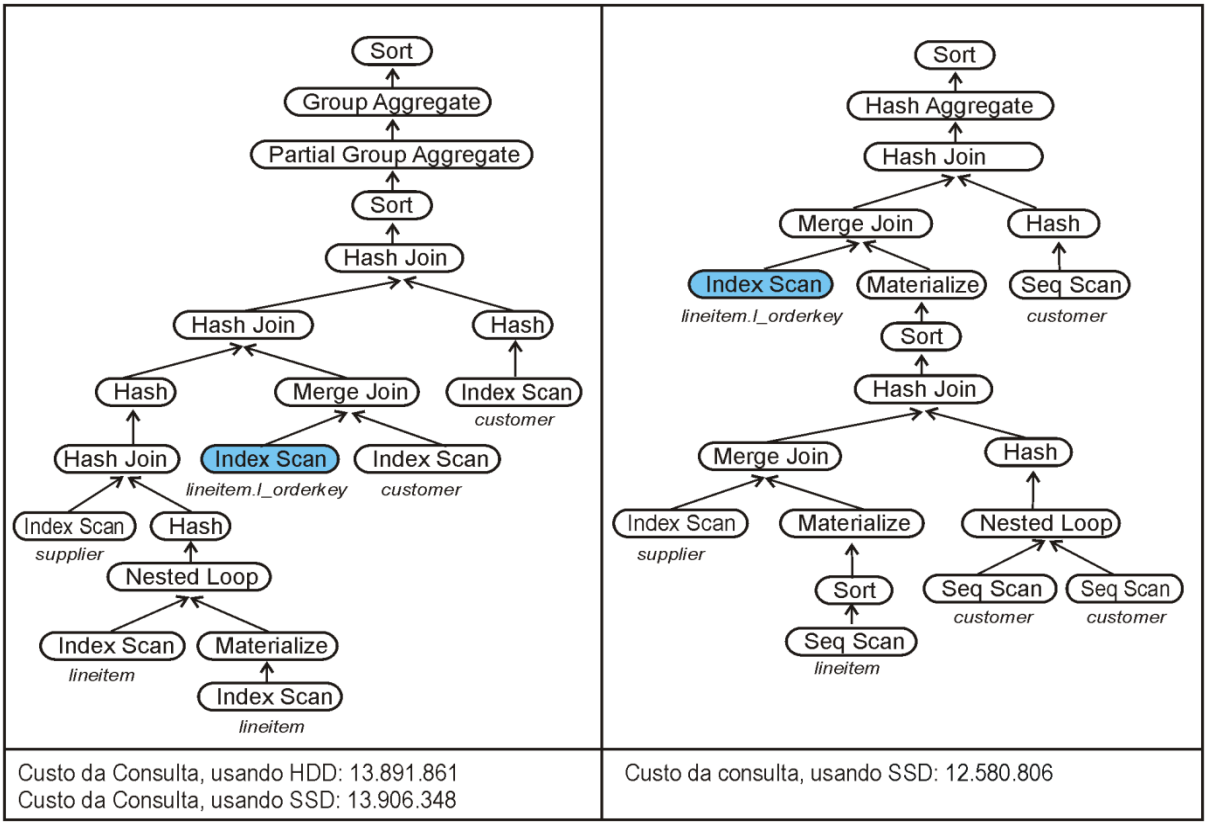
5.6. Análise dos planos de consultas

Para uma análise ampliada dos resultados, foi realizado um processo comparativo entre as mesmas consultas executadas em cenários diferentes, visando avaliar se o plano de execução sofreu alterações devido à mudança de dispositivo de armazenamento.

Nas figuras 5.15 e 5.16, são representados graficamente os planos de consultas de Q5 e Q21 respectivamente, com índices recomendados pelo ITLCS. Analisando a figura 5.15 (a) é possível verificar que o mesmo plano de consulta foi gerado para os cenários com HDD e com SSD utilizando o PostgreSQL padrão. Isto é, foram idênticos o conjunto de algoritmos para o acesso a dados, junções de tabelas e para aplicação de predicados, mesmo que utilizando um cenário com discos SSD, que são mais ágeis.

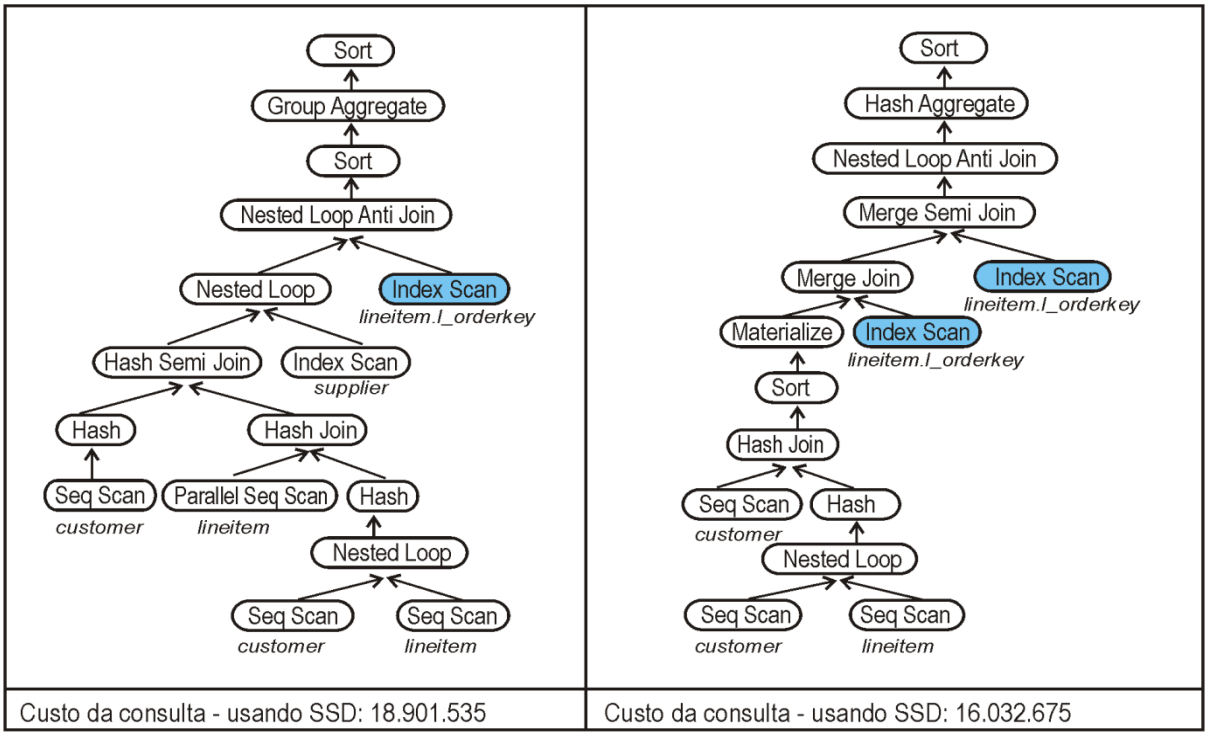
Ainda, observa-se na figura 5.15 (b) que com o PostgreSQL estendido – com novo modelo de custos implementado – foi obtido um plano de consulta diferente do que mostrado na figura 5.15(a) e uma árvore de busca reduzida foi gerada. Além disso, o valor de custo para o cenário usando SSD com PostgreSQL estendido foi menor que para os demais, conforme consta nas representações (a) e (b) da figura 5.15.

É apresentado a seguir o plano da consulta Q21 para os cenários usando SSD com PostgreSQL padrão e com PostgreSQL estendido. É possível verificar na figura 5.16 que em ambos os cenários o ITLCS recomendou o mesmo índice (tabela/coluna: *lineitem/l_orderkey*). Entretanto os planos gerados são diferentes. Observa-se que o plano de consulta da figura 5.16 (b) foi moderadamente mais simplificado, por utilizar um número menor de algoritmos/métodos de acessos; outro ponto a se observar é a utilização do índice secundário em *l_orderkey*, que no plano de consulta da figura 5.16 (b) foi utilizado duas vezes. O plano gerado na figura 5.16 (b), atrelado ao SGBD PostgreSQL estendido – com novo modelo de custos – trouxe benefícios para a consulta Q21, com redução do tempo de execução (Figura 5.6), bem como redução no custo da consulta, se comparado ao custo apresentado na figura 5.16 (a).



(a) Plano de consulta, usando PostgreSQL padrão (b) Plano de consulta usando PostgreSQL estendido.

Figura 5.15. Representação em árvore do plano de execução da consulta Q5.
Fonte: o autor.



(a) Plano de consulta usando PosgreSQL padrão (b) Plano de consulta usando PostgreSQL estendido

Figura 5.16. Representação em árvore do plano de execução da consulta Q21.
Fonte: o autor.

Após avaliações nos planos de consulta de Q5 e Q21, bem como de outras consultas do *benchmark* nos cenários usando HDD e SSD, verificou-se que muitos planos de consulta são idênticos, ainda que alterado o dispositivo de armazenamento. Em linhas gerais, isso evidencia que o SGBD não foi capaz de reconhecer os custos de E/S diferentes entre os dispositivos de armazenamento.

5.7. Estimativa de custo versus tempo de execução de consultas

As estimativas de custo computacional para executar uma consulta, evidenciam-se de suma importância para atividades de *tuning*. São muito empregadas também por rotinas internas dos SGBDs, pois otimizadores baseados em custos as utilizam em comparações de diferentes planos de consultas para, assim, escolherem aquele com menor custo [12]. Ferramentas de auxílio ao DBA também utilizam estimativas de custos em suas heurísticas internas; já os DBAs usam essas métricas sobretudo para verificar como será o desempenho de uma consulta, mesmo antes da sua execução. No PostgreSQL é possível verificar as estimativas de custos de uma consulta através do utilitário *Explain* <consulta>.

Nas figuras 5.17 a 5.19, são apresentados dados percentuais relativos ao desempenho de consultas do TPC-H em três cenários diferentes. São avaliados o tempo de execução e o custo estimado, calculados da seguinte forma:

- Custo: o percentual de melhora ou piora de desempenho (*Pdc*) é calculado considerando o custo da consulta sem índices secundários (custo_sem_idx) e com índices secundários (custo_com_idx), conforme expressão 5.1.

$$\boldsymbol{Pdc} = \frac{100 - (\text{custo\_com\_idx} * 100)}{\text{custo\_sem\_idx}} \qquad \text{(Exp. 5.1)}$$

- Tempo: o percentual de melhora ou piora (*Pdt*) é calculado considerando o tempo da consulta sem índices secundários (tempo_sem_idx) e com índices (tempo_com_idx), conforme expressão 5.2.

$$\boldsymbol{Pdt} = \frac{100 - (\text{tempo\_com\_idx} * 100)}{\text{tempo\_sem\_idx}} \qquad \text{(Exp. 5.2)}$$

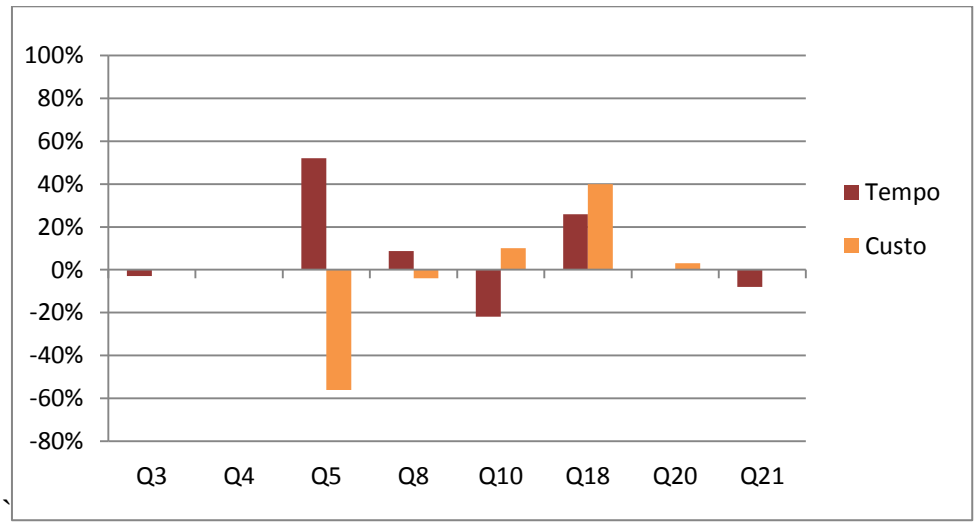


Figura 5.17. Gráfico comparativo das métricas Tempo versus Custo usando HDD e PostgreSQL padrão.
Fonte: o autor.

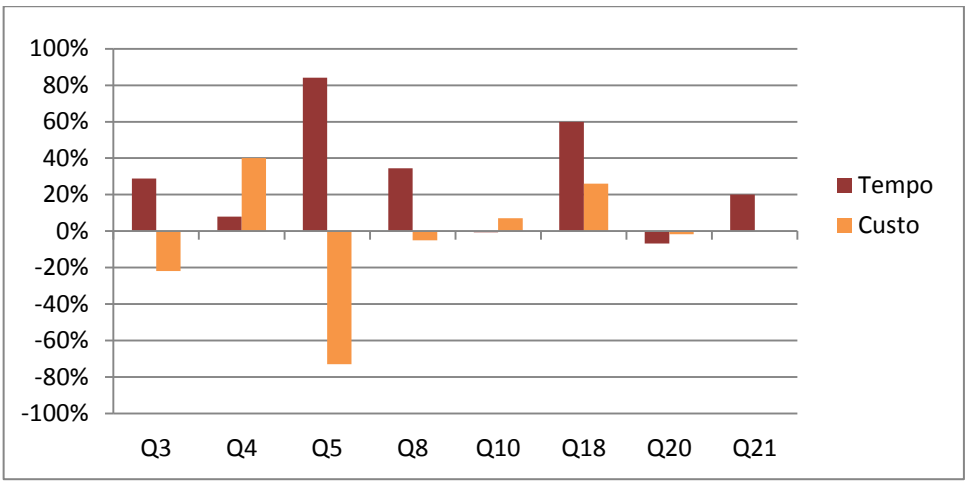


Figura 5.18. Gráfico comparativo das métricas Tempo versus Custo usando SSD e PostgreSQL padrão.
Fonte: o autor.

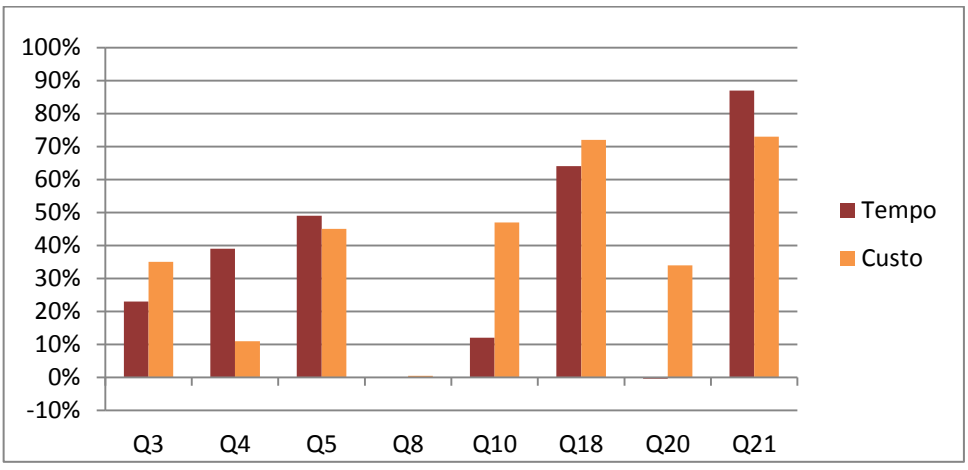


Figura 5.19. Gráfico comparativo das métricas Tempo versus Custo usando SSD e PostgreSQL estendido.
Fonte: o autor.

A comparação de custo versus tempo visa, de modo geral, verificar a sincronia entre essas métricas. Se a métrica de custo tem redução, espera-se, logicamente, que o tempo de execução também tenha redução. Um exemplo disso ocorreu no cenário usando HDD – consulta Q18, figura 5.17 – em que, sem o emprego de índices, a estimativa de custo foi de 39.432.065 e, após a criação de índices o valor da estimativa reduziu em 40%, alcançando 23.555.473. De forma similar, o tempo de execução dessa consulta reduziu em 26%, de 554 segundos para 410 segundos, mostrando sincronia entre as métricas (custo versus tempo), pois, ao se reduzir o custo, o tempo de execução também foi reduzido.

Embora para o exemplo citado (consulta Q18), tenha ocorrido sincronia entre as métricas, para as consultas Q3, Q5, Q8, Q10 e Q21, isso não ocorreu. Um exemplo é a consulta Q5 (figura 5.17), que indicou a piora de 56% no custo estimado com a utilização de índices. No entanto, o tempo de execução da consulta teve redução de mais de 50%, mostrando incoerência entre a estimativa de custos e tempo.

De forma similar, no cenário usando SSDs, apenas as consultas Q4, Q18 e Q20 tiveram sincronia entre as métricas custo versus tempo, como pode ser verificado na figura 5.18.

O erro nas estimativas de custos, em algumas situações, pode ocorrer pelo fato de o SGBD estar com as estatísticas desatualizadas. No entanto, nos experimentos realizados neste trabalho essa hipótese foi descartada, pois as estatísticas são sempre coletadas antes do início do processo e durante o processo não ocorre atualização ou inserção de novos dados nas tabelas.

Um grave problema decorrente do erro nas estimativas de custos diz respeito à atuação dos DBAs e das ferramentas de *tuning*, pois, muitas vezes, as métricas de custos são utilizadas como fator de decisão para ações de *tuning*, e, portanto, estimativas incorretas vindas do SGBD podem causar decisões equivocadas por parte desses profissionais ou dessas ferramentas.

Diferentemente dos cenários que utilizaram o PostgreSQL padrão (figuras 5.17 e 5.18), no cenário que utilizou o SGBD PostgreSQL estendido (figura 5.19) constata-se que, de forma geral, as métricas de custo e tempo tiveram sincronia. Como esperado, o SGBD foi capaz de reconhecer as novas características de custos de E/S dos SSDs com base nas alterações no modelo de custos do SGBD implementadas neste trabalho. A introdução de novos parâmetros para reconhecer leitura e escrita assimétricas resultou em melhor sincronia

entre as estimativas de custos versus tempo real de execução, além de ser possível, desta forma, a obtenção de planos de consultas mais eficientes, como mostrado nas figuras 5.15 (b) e 5.16 (b).

5.8. Características da carga de trabalho versus desempenho

O *benchmark* TPC-H simula o desempenho de um ambiente com características OLAP (*On Line Analytical Processing*) e compreende um conjunto de consultas *ad-hoc*. Considerado por profissionais da área de gerenciamento de dados como um bom projeto de *benchmark,* ele se caracteriza por possuir cargas de trabalho e atualizações que simulam problemas reais encontrados por profissionais de banco de dados. Entre suas características, destacam-se alguns pontos, que, em Boncz et al. [76], são chamados de "pontos de choque". São características da carga de trabalho consideradas problemas tecnológicos derivados do *benchmark,* nos quais ações para resolver estes problemas trarão vantagens significativas para um produto. Os pontos que serão discutidos nesta subseção são: (i) desempenho de operações de Agregação; (ii) desempenho de operações de Junções; e (iii) localidade de acesso a dados.

5.8.1 Desempenho de junções

Junções (*Joins*) representa um dos operadores relacionais mais custosos em consultas SQL. Entre os vários métodos de junções, os mais comuns são baseados em *hash*, índices e *nested loop*. Não é possível afirmar que um método seja sempre superior a outro, pois isso depende muito do ambiente de *hardware/software* e do projeto físico do banco de dados.

Métodos de junções baseados em índices são muito utilizados nas situações em que os dados das chaves de junção estão indexados. Para ambientes com discos HDD, se a chave de junção estiver em um índice clusterizado, isso será vantajoso em relação a um índice não clusterizado, pois HDDs têm maior desempenho em leitura sequencial, devido a sua latência mecânica. Todavia, caso sejam utilizados discos SSD, essa diferença não será significativa pela ausência de latência mecânica dos SSDs, possibilitando que o acesso a um índice não-clusterizado seja próximo ao de um índice clusterizado.

Entre as consultas do TPC-H, as que têm maiores junções são Q9 e Q18, pois envolvem as duas maiores tabelas do *benchmark*: *lineitem* e *orders*. Nos experimentos realizados neste capítulo é possível verificar (tabela 5.1) que na consulta Q9 somente a ferramenta POWA sugeriu um índice, tanto no cenário com HDD quando no cenário com

SSD. No entanto, ao executar a consulta com a presença do índice sugerido, o otimizador de consultas decidiu não utilizá-lo. O fato de o otimizador não utilizar nenhum índice para a consulta Q9 possivelmente vem do fato de a consulta não usar predicados de seleção para a junção das maiores tabelas, além da presença de outras junções com tabelas menores. Isso justifica a escolha por varredura completa, neste caso, já que a maior parte da tabela foi solicitada pela consulta e deve ser recuperada.

De outra forma, na consulta Q18, além da Junção de grandes tabelas, existem também algumas operações de agregação e ordenação. Para esse caso, o ITLCS desenvolvido neste trabalho recomendou um índice para a tabela/coluna *lineitem/l_orderkey* nos três cenários. Além disso, nos cenários usando SSD com PostgreSQL padrão e estendido (figuras 5.6) este índice foi utilizado duas vezes dentro da mesma *query*: (1) na junção das tabelas *lineitem* e *orders*; e (2) na subconsulta interna ao operador IN. Comparando à execução da consulta Q18 nos diferentes cenários (figuras 5.2, 5.4 e 5.6), verifica-se que os dois cenários que utilizaram SSD obtiveram ganho de performance em relação ao método NO-INDEX (sem índices secundários), de 55% e 64%, contra 25% quando utilizando HDD.

Com exceção das consultas Q1 e Q6 todas as outras consultas realizam junções. Mas, diferentemente das consultas Q9 e Q18, elas estão invariavelmente sobre relacionamentos de chave estrangeira N:1 ou 1:N. Além disso, todas as outras consultas envolveram seleções e muito frequentemente o lado :1 da junção é filtrado por predicados. Isso significa que as tuplas do lado N: em vez de encontrarem exatamente um dado-chave de junção, geralmente, não encontram nenhum. Por causa dessa filtragem, o uso de índices para junções nos testes, tanto utilizando HDD quanto SSD, beneficiaram apenas as junções entre grandes volumes de dados como, por exemplo, para a consulta Q18 e Q21, destacando a maior eficiência quando utilizando SSDs.

5.8.2. Desempenho de agregação

Para a maioria das consultas do TPC-H, operações de agregação são comuns, com forte incidência nas consultas: Q1, Q3, Q4, Q10, Q13, Q18, Q20 e Q21. Portanto, para um melhor desempenho geral do TPC-H, operações de agregação devem ter um bom desempenho.

Entre os vários métodos para operações de agregação, destacam-se aqueles baseados em *hash* ou os métodos baseados em índices, implementados em árvore B+.

Hashing fornece o acesso rápido para a recuperação de um registro arbitrário, dado o valor do campo de *hash* e, normalmente, usa uma *hash-table* para armazenar chaves de agrupamento. Uma das principais vantagens de métodos baseados em *hash* é possuir um custo de pesquisa constante. No entanto, quando a quantidade de chaves de agrupamento distintas é grande, pode ocorrer de a *hash-table* não caber nos vários níveis de cache de CPU, ocasionando falhas de cache e tornando o acesso mais dispendioso, pois um espalhamento (*spilling*) de agregação é necessário, no qual inicialmente as tuplas da *hash-partition* vão para arquivos diferentes no disco com base no valor de *hash* e, a posteriori, esses arquivos são agregados dentro da RAM um por vez.

Métodos baseados em índices, implementados em árvore B+, são uma opção aos métodos baseados em hash, por lidarem melhor com o crescimento do volume de chaves de pesquisa, pois nas árvores B+ é possível armazenar milhares de chaves por nó e, em geral, a altura da árvore tende a ser reduzida, mesmo contendo milhões de chaves indexadas. Como a obtenção de cada nó é feita por um acesso a disco, a pesquisa por determinada chave pode ser realizada com poucos acessos a disco. Considerando que o acesso indexado é normalmente dado por um acesso aleatório, a utilização de SSD traz ainda como vantagens o baixo custo de acesso aleatório para acesso aos índices e operações de leitura ágeis.

Considerando o desempenho do ITLCS relativo às consultas com forte agregação, na tabela 5.1, verifica-se que, no ambiente utilizando HDD foi indicada a criação de índices para as consultas Q18, Q20 e Q21, mas como pode ser observado, somente a consulta Q18 obteve redução no tempo de execução. No cenário utilizando HDD, o maior problema foi na consulta Q21, por ocorrência de subconsultas nas cláusulas *exists* e *not exists*, manuseando dados da maior tabela do *benchmark*, que fizeram o tempo piorar em 6% em comparação ao cenário utilizando índices primários, muito provavelmente pelo alto custo de acesso aleatório dos HDDs.

Para o ambiente utilizando SSD, o ITLCS criou índices para as consultas Q3, Q4, Q8, Q18, Q20 e Q21. Como visualizado na tabela 5.1, o tempo de execução teve redução para a maioria das consultas – Q3, Q4, Q18 e Q21 – mostrando as vantagens de se utilizar índices para operações de agregação quando utilizando SSDs em relação ao uso HDDs.

5.8.3. Localidade de acesso a dados

Um recurso importante utilizado por DBAs para acelerar a execução de consultas, sobretudo em ambientes usando HDD, é priorizar métodos que recuperem dados que estejam fisicamente próximos. Uma técnica bastante popular para beneficiar a localidade dos dados é a utilização de índices clusterizados. A criação de índice clusterizado pode ser vantajosa especificamente para ambientes com HDD, mas uma questão importante é decidir qual coluna usar como chave, haja vista que há restrição de somente um índice clusterizado por tabela.

No TPC-H, uma oportunidade para aproveitar a localidade dos dados por meio de índices clusterizados é sobre as consultas que envolvem as duas maiores tabelas (*lineitem* e *orders*) contendo filtros de intervalo de datas. Na tabela *orders*, existe somente *o_orderdate* para ser escolhida para ser indexada. Na tabela *lineitem*, a coluna *l_shipdate* é usada com mais frequência (em Q6, Q15 e Q20) do que a *l_receiptdate* (apenas em Q12).

Um exemplo de junção entre *lineitem* e *orders* que poderia levar à alta localidade nos dados seria, por exemplo, a criação de um índice clusterizado na Tabela *orders*, na coluna *o_orderdate* e, na tabela *lineitem*, um índice clusterizado em *l_shipdate*; em combinação com um índice não clusterizado para as chaves primárias.

Nos discos SSD técnicas que priorizem a localidade de dados, como, por exemplo, os índices clusterizados, não trazem grandes benefícios se comparados aos obtidos em ambientes com HDDs, pelo simples fato de que os SSDs não são afetados pela latência mecânica, e o tempo de espera para acessos aleatórios é levemente superior ao acesso sequencial.

Nos experimentos realizados, destacam-se as consultas Q3, Q10, Q14 e Q20, que utilizaram índices não clusterizados, mesmo assim obtiveram significativo ganho de desempenho, conforme apresentado na tabela 5.1, evidenciando que o armazenamento em SSD não é sensível à localidade dos dados.

5.9. Discussão dos Resultados

5.9.1. Desempenho do ITLCS na recomendação de índices

Após os experimentos empíricos utilizando o ITLCS em comparação às ferramentas para *tuning* de índices EDB e POWA, verificou-se que o ITLCS obteve o melhor desempenho no geral, mesmo tendo o menor número de índices criados. Isso traz como vantagens diretas:

(i) economia de espaço de armazenamento, evitando a criação de índices não utilizados, como ocorreu principalmente com o método POWA; (ii) menor *overhead* na atualização do índice, pois devido ao seu menor número de índice, também será menor o seu número de atualizações, para cada *insert*/*update* realizados nas tabelas de origem.

Destaca-se que o desempenho mais significativo obtido pelo ITLCS foi no cenário usando SSD com o PosgreSQL estendido, implementado neste trabalho. Como esperado, com a geração de estimativas de custos mais assertivas, o ITLCS conseguiu encontrar um conjunto de índices diferente de outros cenários, bem como planos de consulta com estratégias que resultaram em diminuição do tempo de execução da carga de trabalho.

5.9.2. Regras geradas

Analisando o conjunto das melhores regras encontradas pelo ITLCS relativo a quantidade de tuplas e a seletividade, conforme reportado nas tabelas 5.3 e 5.4 respectivamente, observa-se que as regras finais direcionadas a SSD apresentaram significativa diferenciação em relação às regras para HDD, demonstrando a capacidade do ITLCS em adaptadar regras ao ambiente em que atua.

Vale lembrar que, inicialmente, o ITLCS parte de um conjunto de regras direcionado para HDD, idêntica as regras direcionadas para SSD, e devido ao processo evolutivo para descoberta de novas regras, essas são adaptadas ao ambiente. Desta forma, verificada a diferenciação das regras utilizadas no ambiente HDD e SSD, e a melhora de desempenho a ela relacionada, comprova-se a primeira hipótese desta tese, de que o armazenamento em SSD pode se beneficiar do ajuste das regras de tuning originalmente elaboradas para HDD, visando maximizar o desempenho.

5.9.3 Planos de consultas

A seção 5.6 identificou que, utilizando o SGBD PostgreSQL padrão os planos de consultas gerados para ambientes com HDD e SSD foram idênticos em várias consultas, mesmo que utilizando dispositivos mais ágeis, como os SSDs. Destaca-se a necessidade de alterações no modelo de custos dos SGBDs, como mencionado na segunda hipótese de pesquisa, definida no capítulo 1 – seção 1.2.

Como pode ser visto anteriormente nas figuras 5.15 e 5.16, se comparados os planos de consultas de Q5 e Q21 usando o PostgreSQL padrão e estendido, verifica-se que os planos

de consultas gerados pela versão estendida obtiveram menor custo e menor tempo de execução, melhorando o desempenho dessas consultas. Isso comprova a hipótese lançada, segundo a qual os SGBDs que utilizem armazenamento híbrido devem ter seu modelo de estimativa de custos alterado para maximizar o seu desempenho.

5.9.4. Estimativa de custo versus tempo de execução de consultas

Estimar o custo computacional para a execução de uma consulta é de suma importância para atividades de *tuning*. No entanto, conforme análise descrita na seção 5.7, foram identificados problemas nas estimativas de custos utilizando o SGBD PostgreSQL padrão. Vários exemplos demonstraram que o custo de uma consulta com índice seria maior do que sem utilizá-lo. No entanto, ao executar a consulta usando o referido índice, ocorreu redução no tempo de execução, indicando claramente um erro na estimativa de custo.

Para o cenário que utilizou o PostgreSQL estendido (figura 5.19), constata-se que, de modo geral, as métricas de custo e tempo tiveram sincronia, demonstrando a relevância das alterações aqui implementadas, no modelo de custos do SGBD PostgreSQL – propostas no capítulo 4 seção 4.8. Desta forma, evidencia-se que o SGBD foi capaz de reconhecer as novas características de custos de E/S dos SSDs, o que resultou em melhores estimativas e que, por sua vez, acarretou maior sincronia entre as estimativas (custos x tempo real de execução). Este resultado, assim como na subseção anterior (subseção 5.9.3), corrobora a segunda hipótese de pesquisa, da necessidade em modificar internamente os SGBDs.

Capítulo 6

Conclusão

Apresentam-se neste capítulo as conclusões da pesquisa desenvolvida, suas contribuições, sua relevância, as limitações verificadas em seu percurso e possibilidades de trabalhos futuros.

As aplicações de bancos de dados têm se tornado cada vez mais complexas e variadas e podem ser caracterizadas por seu grande volume de dados e por sua elevada demanda quanto à redução no tempo de resposta das consultas e à vazão (*throughput*) das transações. Neste contexto, ações de *tuning*, sobretudo relacionadas a índices, têm se revelado importantes, porque influem diretamente no desempenho dos sistemas de banco de dados.

Com o advento dos SSDs, uma revolução ocorreu na tecnologia de armazenamento, trazendo consigo um novo desafio tanto para DBAs quanto para as ferramentas de auxílio ao DBA e, também, para os SGBDs e seus modelos de custos.

Nesta tese o objetivo principal foi o desenvolvimento e avaliação de um algoritmo responsável por realizar o *tuning* de índices em SGBDRs que utilizam armazenamento secundário híbrido (HDD e SSD).

6.1. Análise das principais contribuições

As principais contribuições da pesquisa podem ser resumidas como:

a) caracterização dos dispositivos SSDs e experimentos empíricos para demonstrar que os SGBDRs e ferramentas de auxílio ao DBA não conseguem identificar os diferentes custos entre HDD e SSD;
b) desenvolvimento e avaliação de um algoritmo para *tuning* de índices usando o modelo de SC;
c) análise das diferenças entre regras especializadas em HDD e SSD, geradas pelo SC;
d) criação de extensões no SGBD PostgreSQL, responsáveis por modificar seu modelo de custos, para reconhecimento de leitura e escrita assimétricas, juntamente com o desenvolvimento de um algoritmo de calibração para busca de coeficientes ótimos para parâmetros de custos do SGBD;
e) análise dos resultados por diferentes abordagens como: (i) comparação dos planos de consultas gerados entre diferentes cenários; (ii) análise da sintonia entre estimativas de custo e tempo; e (iii) análise de desempenho ampliada em operações de agregação, junção e localidade de acesso a dados.

6.1.1. Caracterização dos SSDs e experimentos usando HDD e SSD

Inicialmente, foram caracterizadas as novas propriedades dos SSDs. Verificou-se que os SGBDRs e as ferramentas de auxílio ao DBA não diferenciam as características de E/S exclusivas entre o HDD e SSD. Foi demonstrado empiricamente que, considerando que a mesma consulta é executada em diferentes dispositivos de armazenamento, na maioria dos casos os SGBDs geram os mesmos planos de consulta, mesmo que discos mais rápidos estejam disponíveis. Nota-se que nesses experimentos o tempo de execução com SSD em geral é menor do que usando o HDD, mesmo que os planos de consulta sejam equivalentes, ou seja, utilizando os mesmos métodos de acesso a dados.

6.1.2. Um algoritmo para *tuning* de índices utilizando sistemas classificadores

Foram apresentados os desenvolvimentos metodológicos necessários para uso do modelo de SC a ser aplicado em *tuning* de índices, no qual se destacam as seguintes inovações:

a) capacidade do SC de diferenciar custos de E/S entre HDD e SSD;
b) disponibilidade de várias ações de *tuning* em uma única ferramenta, incluindo a criação, remoção, reconstrução (se fragmentado) e a migração de índices (de HDD para SSD), destacando que ações de reconstrução e migração não foram encontradas nas ferramentas avaliadas ou em outras disponíveis para o SGBD utilizado;

c) geração de um conjunto de regras compreensíveis, permitindo assim que pesquisadores e profissionais da área de banco de dados consigam analisar e confiar nas ações sugeridas pelo ITLCS;
d) capacidade de descobrir novas regras de *tuning* especializadas, conforme o tipo de armazenamento secundário (HDD/SSD), importante para o processo de adaptação do ITLCS às mudanças no SGBD ou na carga de trabalho.

Apesar do SC ter sido projetado somente para diferenciar ações entre HDD e SSD, caso haja intenção de se personalizar o *tuning* para outras tecnologias de armazenamento que existam ou que venham a surgir, as alterações são possíveis. Basicamente, ajustes desse gênero consistem em adicionar tipos de regras ao SC e alterar sua heurística de benefícios (HRPC), possibilitando ao SC personalizar o *tuning* para os dispositivos, sejam eles HDDs, SSDs, ou outros.

Salienta-se também que o algoritmo proposto é a primeira abordagem para *tuning* de índices que utiliza como modelo um SC, o qual trouxe como vantagem a obtenção de um conjunto de regras compreensíveis o bastante para que pesquisadores e profissionais da área de banco de dados consigam analisar e confiar nas ações sugeridas pelo ITLCS. Além disso, ao analisar os experimentos realizados (Capítulo 5) com outras ferramentas para *tuning* de índices utilizadas por DBAs, observa-se que o ITLCS obteve melhores resultados na maior parte dos casos.

Ainda verificou-se que, em geral, ferramentas comerciais ou *open-sources* usadas por DBAs tendem a se concentrar na otimização de operações de localização de registros de dados e, muitas vezes, preocupam-se somente em situações com maior potencial de melhoria. Todavia, abordagens baseadas em algoritmos bioinspirados, como os utilizados nesta tese, operam em um espaço de busca, onde todas as possibilidades são consideradas, o que contribui para que a solução final não seja limitada a mínimos ou máximos locais.

6.1.3. Análise das regras aplicadas a HDD e a SSD

Como o ITLCS é um algoritmo baseado em regras, a análise do seu conjunto de regras (classificadores) é importante. Para isso, foram realizados experimentos aplicando regras em ambiente com HDD e SSD, visando verificar o comportamento das regras nesses ambientes. Conforme resultados apresentados no capítulo 5 – seção 5.5 comprova-se a primeira hipótese da pesquisa: adaptando-se as regras de *tuning* conforme o tipo de armazenamento, acarreta em melhora de desempenho do sistema.

6.1.4. Extensões no modelo de custos do SGBD PostgreSQL

Foram apresentadas as principais características do modelo de custos proposto por Bausch et al. [15] projetado para o PostgreSQL 9.0.1, o qual foi adaptado para a versão 10.1 do PostgreSQL e utilizado em um dos cenários de teste. As alterações internas foram importantes para que o SGBD conseguisse diferenciar os custos de E/S de acessos sequenciais e aleatórios em operações de leitura e escrita. Para isso, foram necessárias inserções de novos parâmetros no SGBD em substituição a outros. Também foram modificadas as funções de custos dos principais algoritmos de acesso a dados, como: *external sort, hash join, sequential scan, index scan e parallel index scan* – este último presente somente a partir da versão 10.0 do PostgreSQL.

Para escolher um conjunto adequado de coeficientes para os parâmetros relacionados ao custo de leitura e escrita do SGBD, um algoritmo foi proposto usando a meta-heurística algoritmo genético. A utilização do AG desenvolvido foi importante sobretudo para os novos parâmetros do modelo de custos implementado.

Experimentos utilizando o SGBD PostgreSQL estendido demonstraram expressiva melhora de desempenho influenciadas pelo novo modelo de custos. Merece destaque a consulta Q21, que, apesar de ser a mais custosa e demorada do *benchmark*, teve uma redução no tempo de 88% no ambiente com versão estendida contra 19% obtidos utilizando a versão padrão do SGBD.

Outra vantagem observada por meio do SGBD PostgreSQL estendido é a sintonia entre as estimativas de custos e tempo, conforme indicam as figuras 5.17, 5.18 e 5.19. Isto é, não ocorreram grandes discrepâncias como nos outros cenários em que, para alguns casos, as estimativas de custos sob a adoção de índices indicava queda no desempenho e, ao executar a consulta, observou-se alta no desempenho. Foi possível, também, verificar que em determinados casos o SGBD gerou planos de consultas mais eficientes, como evidenciado nas Figuras 5.15 (b) e 5.16 (b).

Destaca-se ainda que, considerando que o ITLCS é um algoritmo externo ao SGBD, portanto não intrusivo, existe a possibilidade de adaptá-lo para uso em outros SGBDRs, exigindo poucos ajustes.

6.1.5. Análise dos resultados em diferentes abordagens

Experimentos empíricos utilizando o *benchmark* TPC-H permitiram avaliar o algoritmo ITLCS em três cenários diferentes, possibilitando a análise de desempenho com ou sem índices em

diferentes casos: variando os tipos de armazenamento (HDD e SSD); e usando duas versões do SGBD PostgreSQL (versão padrão e versão estendida).

Também foram realizadas análises dos resultados em diferentes abordagens:

a) comparação dos planos de consultas gerados entre diferentes cenários, verificando se ocorreu alta ou baixa no desempenho, após a troca de dispositivos de armazenamento e/ou alterações da versão do SGBDR (PostgreSQL versão padrão e versão estendida);
b) análise da sintonia entre as estimativas de custo e tempo, visando verificar o comportamento geral destas, pois se ocorrer, por exemplo, alta de desempenho para uma métrica, espera-se que, da mesma forma, ocorra alta de desempenho para a outra métrica; e
c) análise de desempenho obtido no *benchmark* utilizando HDD e SSD, sob aspectos específicos da carga de trabalho, como desempenho de operações de agregação, desempenho de operações de junção, e aspectos da localidade de acesso a dados.

Os cenários de teste utilizados foram de fundamental importância, pois permitiram verificar como foi o desempenho do ITLCS utilizando HDD e SSD, e também usando o SGBD PostgreSQL em versões padrão e estendida. Além disso, foi possível comparar o desempenho do ITLCS frente às ferramentas para tuning: POWA e EDB, evidenciando a superioridade de desempenho quando utilizando o ITLCS.

Os experimentos também mostraram que no cenário com índices gerados pelo ITLCS, utilizando armazenamento SSD e PostgreSQL estendido, foram obtidos os melhores resultados entre todos os cenários. Isso ocorreu devido às melhores estimativas de custos geradas pelo SGBD, o que, por consequência gerou melhores planos de consultas. As ações de *tuning* de índices do ITLCS também foram beneficiadas por melhores estimativas de custos do SGBDR, pois internamente o ITLCS utiliza essas estimativas para realizar a seleção de índices e nas suas heurísticas de punição e recompensa de classificadores (HRPC).

6.2. Limitações desta tese e possibilidades de trabalhos futuros

Durante o desenvolvimento desta tese, foram identificadas algumas limitações do algoritmo ITLCS que podem ser objeto de estudos para futuras pesquisas e que, hipoteticamente, poderiam gerar resultados ainda melhores.

O ITLCS foi projetado para trabalhar apenas com índices secundários de coluna única. No entanto, existem situações que um único índice de múltiplas colunas pode trazer maiores benefícios do que vários índices de coluna única, além de demandar um espaço menor.

Outra limitação do trabalho é a ausência da ação de exclusão de índices. A exclusão de índices pouco utilizados é importante, pois elimina operações de atualizações em índices pouco usados, além de liberar espaço em disco. Essa ação, porém, deve ser avaliada com cuidado, porque um índice não utilizado em um período poderá ser útil num período futuro e, desta forma, essa intervenção deve ter critérios bem definidas.

Outro aspecto que se observou como limitante foi em relação ao modelo de indivíduo (Capítulo 4, Seção 4.2). Verificou-se que o domínio de alguns genes (G) foi superestimado, como por exemplo, os genes G10 e G12. Como reportado nas Tabelas 5.3 e 5.4, verifica-se que somente três faixas de valores foram utilizadas para o domínio definido. Com a readequação destes domínios, o processo de evolução das regras pode ser beneficiado, por não gerar regras com valores que nunca serão úteis para o sistema.

Como projeção de trabalhos futuros a ser realizados com base nesta pesquisa, tem-se, por exemplo, a inclusão de novas métricas de benefícios para as estruturas criadas/sugeridas. Além disso, sugere-se a inclusão de outras ações de *tuning*, tais como seleção, criação e/ou manutenção de: índices *hash*, visões materializadas, partições de tabelas e histogramas. Salienta-se que, para estabelecer essas novas ações, serão necessárias tanto alterações no indivíduo, a fim de cobri-las, quanto na função de avaliação do ITLCS.

Outra sugestão para trabalho futuro seria investigar e propor técnicas que potencializam e aproveitam os recursos de paralelismo dos SSDs, atrelados ao emprego de cargas de trabalhos que simulem acessos simultâneos.

Também seria possível um trabalho futuro que busque a realização de experimentos com *layouts* diversos de armazenamento (RAID, SAS). Isso permitiria avaliar o emprego do ITLCS nesses diferentes cenários, bem como ampliar o seu uso.

6.3. Publicações Relacionadas

Com base nos resultados desta pesquisa, alguns artigos científicos estão em processo de escrita para posterior submissão a periódicos e conferências. Os artigos que já foram objetos de publicações ou aceitos para publicação são os seguintes:

- PEDROZO, Wendel Góes. Index Self-tuning in Database Management Systems using Learning Classifier Systems. Doctoral Consortium. In: *41st European Conference on Information Retrieval (ECIR'19)* 14–18 Abril 2019, Cologne, Germany. (artigo aceito para publicação – Qualis A2)

- PEDROZO, Wendel Góes; NIEVOLA, Júlio Cesar; RIBEIRO, Deborah Carvalho. An Adaptive Approach for Index Tuning with Learning Classifier Systems on Hybrid Storage Environments. In: *20th International Conference on Hybrid Artificial Intelligence Systems (HAIS'18).* Oviedo, Spain, 20–22 jun. 2018 (Qualis B1).

- PEDROZO, Wendel Góes; NIEVOLA, Júlio Cesar; RIBEIRO, Deborah Carvalho. An Adaptive Approach for Index Tuning with Learning Classifier Systems on Hybrid Storage Environments. J. de Cos Juez et al. (Eds). Springer, Cham, 2018: Lecture Notes in Computer Science, vol 10870. p. 716-729 (Qualis C).

- PEDROZO, Wendel Goes; NIEVOLA, Júlio Cesar; RIBEIRO, Deborah Carvalho. Gerenciamento autônomo de projeto físico de banco de dados. In: *XXII SEMIC. Seminário de Iniciação Científica. 3º Congresso Sul Brasileiro de Iniciação Científica & Pós-Graduação*, 2014, Curitiba. Caderno de Resumos do XXII Seminário de Iniciação Científica. Curitiba: Champagnat, 2014.

REFERÊNCIAS

[1] EVANGELISTA, N. L.; FILHO, J. A. Bt-Join: A Join Operator for Asymmetric Storage Device. In: *ACM Symposium on Applied Computing (SAC)*, Salamanca, Spain, p. 988-993, 2015.

[2] RAMAKRISHNAN, R.; GEHRKE, J. Sistemas de Gerenciamento de Banco de Dados. 3.ed. São Paulo: McGraw Hill, 2008.

[3] CHAUDHURI, S.; DATAR, M. ; NARASAYYA, V.. Index selection for databases: A hardness study and a principled heuristic solution. IEEE Transactions on Knowledge and Data Engineering, 16(11):1313–1323, 2004.

[4] LUHRING, M.; SATTLER, K.; SCHMIDT, K.; SCHALLEHN, E.: Autonomous Management of Soft Indexes. In: *IEEE International Conference on Data Engineering Workshop (ICDE)*, Istanbul, Turkey, p. 450-458, 2007.

[5] WANG, L.; WANG, H. A New Self-adaptive Extendible Hash Index for Flash-based DBMS. In: *IEEE International Conference on Information and Automation (ICIA)*, p. 2519-2524, 2010, Harbin, China.

[6] LUO, T. et al., 2012. hStorage-DB: Heterogeneity-aware Data Management to Exploit the Full Capability of Hybrid Storage Systems, In: *Proceedings of the Very Large Databases*, Istanbul, Turkey, 2012.

[7] SCHNAITTER, K.; POLYZOTIS, N. Semi-Automatic Index Tuning: Keeping DBAs in the loop. VLDB Endowment, Istanbul, Turkey. v. 5, n.5, p. 478-489, 2012.

[8] JIMENEZ, I.; SANCHEZ, H.; TRAN, Q. T.; POLYZOTIS, N. Kaizen: A Semi-Automatic Index Advisor. In: *ACM SIGMOD International Conference on Management of Data*, Scottsdale, Arizona, USA, p. 685-688, 2012.

[9] ALMEIDA, A. C. ; BRAYNER, A. ; MONTEIRO, J. M. ; LIFSCHITZ, S. ; OLIVEIRA, R. P. Automatic Physical Design Tuning based on Hypothetical Plans. In: *Simpósio Brasileiro de Bancos de Dados (SBBD)*, Salvador, Brasil, p. 115-120, 2016.

[10] LOHMAN, G.; VALENTIN, G.; ZILIO, D.; ZULIANI, M.; SKELLEY, A. DB2 Advisor: An Optimizer Smart Enough to Recommend its Own Indexes. In: *IEEE International Conference on Data Engineering (ICDE)*, 2000, San Diego, CA, USA. p. 101-110, 2000.

[11] MONTEIRO, J. M. F. Uma Abordagem Não-Intrusiva para a Manutenção Automática do Projeto Físico de Bancos de Dados. 2008. 201 f. Tese (Doutorado em Informática) – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUC-RIO), Rio de Janeiro, 2008.

[12] ELMASRI, R.; NAVATHE, S. B. Sistemas de Banco de Dados. 4. ed. São Paulo: Addison Wesley, 2005.

[13] CHAN, I.; ASHDOWN., L. Oracle Database Performance Tuning Guide, 11g Release 2. Redwood City: Oracle, 2010

[14] LEE, E.; LEE, S.; PARK, S. Optimizing Index Scans on Flash Memory SSDs. SIGMOD Record, v. 40, n. 4, p. 5-10, 2011.

[15] BAUSCH, D.; PETROV, L.; BUCHMANN, A. Making Cost-Based Query Optimization Asymmetry-Aware. In: *International Workshop on Data Management on New Hardware (DAMON)*, 2012, Scottsdale, USA, p. 24-32, 2012.

[16] Enterprise DB Index Advisor (2018).<
https://www.enterprisedb.com/docs/en/10.0/EPAS_Guide_v10/EDB_Postgres_Adva nced_ Server_Guide.1.34.html>

[17] POWA. (2018) Postgresql Workload Analyser. <http://powa.readthedocs.io>

[18] FINKELSTEIN, S.; SCHKOLNICK, M.; TIBERIO, P. Physical data-base design for relational databases. ACM Transactions on Data-base Systems (TODS), v. 13, n. 1, p. 91-28, 1988.

[19] URBANOWICZ, R. J.; MOORE, J. H. Learning classifier systems: a complete introduction, review, and roadmap. J. Artif. Evol. App., Hindawi Publishing Corp., New York, NY, USA, v. 2009, p. 1:1–1:25, 2009. ISSN 1687-6229.

[20] POKORNY, J. Nosql databases: a step to database scalability in web environment. In: *International Conference on Information Integration and Web-based Applications and Services (iiWAS)*, New York, NY, USA. p. 278-283, 2011.

[21] GARCIA-MOLINA, H.; ULLMAN, J., WIDOM, J.. Implementação de Sistemas de Bancos de Dados. Rio de Janeiro: Campus, 2001.

[22] CODD., E. F. A Relational Model of Data for Large Shared Data Banks. Commun. ACM. 13 (6):377-387, 1970.

[23] IOANNIDIS, Y. E.; QUERY OPTIMIZATION. ACM COMPUT. SURV., 28(1):121–123, 1996.

[24] SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. Sistema de Banco de Dados. Rio de Janeiro: Campus, 2006.

[25] LIGHTSTONE, S. S.; LOHMAN, G. ; ZILIO, D.. Toward autonomic computing with db2 universal database. SIGMOD Rec., 31(3):55–61, 2002.

[26] MAIER, C., DASH, D., ALAGIANNIS, I., AILAMAKI, A. HEINIS, T. PARINDA: An Interactive Physical Designer for PostgreSQL. In: *International Conference on Extending Database Technology (EDBT)*, Lausanne, Switzerland, 2010.

[27] COSTA, R. L. C.; LIFSCHITZ, S.; NORONHA, M.; SALLES, M. V. Implementation of an agent architecture for automated index tuning. In: *IEEE International Conference on Data Engineering* Workshops, 2005.

[28] REZENDE, S. O. Sistemas inteligentes – fundamentos e aplicações. Editora Manole, 2002.

[29] DAGEVILLE, B.; DIAS, K. Oracle's Self-Tuning Architecture and Solutions. IEEE Bulletin of Technical Commitee on Data Engineering, n. 29, p. 24–31, 2006.

[30] DEJONG, K. A. Analysis of the Behavior of a Class of Genetic Adaptive Systems. 1975. 271 f. Dissertation (Doctor of Philosophy) – Computer and Comunication Sciences, University of Michigan, Hampton, Virginia, 1975.

[31] DEJONG, K. A. Learning with Genetic Algorithms: An Overview. Machine Learning, v. 3, p. 121-138, 1988.

[32] DIAS, K.; RAMACHER, M.; SHAFT, U.; VENKATARAMANI, V.; WOOD, G. Automatic Performance Diagnosis and Tuning in Oracle. In: *Conference on innovative Data Systems Research (CIDR)*, Asilomar, Canadá., p. 1-11, 2005.

[33] OSDL (2018) <https://github.com/OSDLabs>

[34] DUDA, R.O.; HART, P.E.; STORK, D.G.: Pattern Classification, 2ª edição, Wiley-Interscience, 2000.

[35] BRUNO, N.; CHAUDHURI, S. An Online Approach to Physical Design Tuning. In: *IEEE International Conference on Data Engineering(ICDE)*, 2007, Istanbul, Turquia, p. 1-12, 2007.

[36] AGRAWAL, S.; CHAUDHURI, S.; KOLLAR, L.; MARATHE, A.; NARASAYYA, V.; SYAMALA, M. Database Tuning Advisor for Microsoft SQL Server 2005. In: *International Conference on Very Large Databases (VLDB)*, 2004, Toronto, Canadá, p.1-12, 2004.

[37] BERNADO-MANSILLA, E.; GARRELL, J. M. Accuracy-Based Learning Classifier Systems: Models, Analysis and Applications to Classification Tasks. Evolutionary Computation, v. 11, n. 3, p. 209–238, 2003.

[38] GRAEFE, G. The five-minute rule twenty years later, and how flash memory changes the rules. In: *International Workshop on Data Management on New Hardware (DAMON)*, Scottsdale, USA. p. 1-9, 2007.

[39] FREITAS, A. A. A survey of evolutionary algorithms for data mining and knowledge Discovery Advances in Evolutionary Computation, p. 819–845, 2002.

[40] SQL Access Advisor <https://www.oracle.com/technetwork/articles/sql/11g-sqlaccessadvisor-084929.html>

[41] COSTA, R.L.C., LIFSCHITZ, S.: Index self-tuning and agent based databases. In: *Latin-American Conference on Informatics (CLEI)*, Montevideu, Uruguay, p. 1-12, 2002.

[42] HYPOPG: Hypothetical Indexes Support PostgreSQL. (2018) <http://dalibo.github.io/hypopg>

[43] GHODSNIA, P.; BOWMAN, I.; NICA, A. Parallel I/O Aware Query Optimization. In: *ACM SIGMOD International Conference on Management of Data*, Snowbird, UT, USA, p. 349-360, 2014.

[44] GOLDBERG, D. E. Genetic Algorithms in Search, Optimization and Machine Learning. New York: Addison-Wesley, 1989.

[45] BRUNO, N.; CHAUDHURI, S. Automatic Physical Database Tuning: a Relaxation-Based Approach. In: *ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, p. 227–238, 2005.

[46] AILAMAKI, A.; PAPADOMANOLAKIS, S. An Integer Linear Programming Approach to Database Design. In: *International Workshop on Self-Managing Database Systems (SMDB)*, 2007.

[47] HOLLAND, J. H. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. Cambridge, USA: Ann Arbor MIT Press, 1975.

[48] HOLLAND, J. H.; REITMAN, J. S. Cognitive Systems Based on Adaptive Algorithms. In: WATERMAN, D. A.; HAYES-ROTH, F. (Eds.). Pattern-Directed Inference Systems. New York: Academic Press, 1978.

[49] Zilio, D. C., Rao, J., Lightstone, S., Lohman, G. M. Storm, A. J., Garcia-Arellano, C., Fadden, S. DB2 Design Advisor: Integrated Automatic Physical Database Design. In: *International Conference on Very Large Databases (VLDB)*, Toronto, Canadá.. p.1-12, 2004.

[50] CHAUDHURI, S.; NARASAYYA, V. Microsoft Index Tuning Wizard for SQL Server 7.0. In: *ACM SIGMOD International Conference on Management of Data*, Seattle, USA. p. 553-554, 1998.

[51] JONG, K. A. D.; SPEARS, W. M. Learning Concept Classification Rules Using Genetic Algorithms. In: *International Joint Conference on Artificial Intelligence*, p. 651–656, 1991.

[52] BRUNO, N. Automated Physical Database Design and Tuning. Emerging directions in database systems and applications. CRC Press, 2011.

[53] SHARMA, A.; SCHUHKNECHT, F. M.; DITTRICH, J. The Case for Automatic Database Administration using Deep Reinforcement Learning. arXiv preprint arXiv:1801.05643, 2018.

[54] WILSON, S. W. Classifier Fitness Based on Accuracy. Evolutionary Computation, v. 3, n. 2, p. 149–175, 1995.
.
[55] BUTZ, M. V. Rule-based Evolutionary Online Learning Systems: Learning Bounds, Classification, and Prediction. Tese de Doutorado, Graduate College of the University of Illinois at Urbana-Champaign, Urbana, IL, 2004.

[56] WEIKUM, G.; HASSE, C. MONKEBERG, A.; ZABBACK, P. The Comfort Automatic Tuning Project, Invited Project Review. Information Systems, v. 19, n. 5, p. 381–432, 1994.

[57] LIFSCHITZ, S.; MORELLI, E. T. Towards autonomic index maintenance. In: : *Simpósio Brasileiro de Bancos de Dados (SBBD)*, Florianópolis, Brasil, p. 176-290, 2006.

[58] GRAY, J.; FITZGERALD, B. Flash disk opportunity for server applications. Queue, v. 6, n. 4, p. 18-23, 2008.

[59] SMITH, R. E.; GOLDBERG, D. E. Reinforcement Learning with Classifier Systems: Adaptive Default Hierarchy Formation. Applied Artificial Intelligence, v. 6, p. 79–102, 1992.

[60] COMER, D. The Difficulty of Optimum Index Selection. ACM Transactions on Database Systems (TODS), v. 3, n. 4, p. 440-445, 1978.

[61] DIRIK, C., JACOB, B.: The performance of PC solid-state disks (SSDs) as a function of bandwidth, concurrency, device, architecture, and system organization. In: *IEEE Symp. Computer Architecture (ISCA)*, p. 279–289, 2009.

[62] SCHNAITTER, K.; ABITEBOUL, S.; MILO, T.; POLYZOTIS, N. On-line index selection for shifting workloads. IN: *IEEE International Workshop on Self-Managing Database Systems (SMDB)*, Istanbul, Turkey, p. 459–468, 2007.

[63] BOOKER, L. B.; GOLDBERG, D. E.; HOLLAND, J. H. Classifier Systems and Genetic Algorithms. Journal Artificial Intelligence, Amsterdam, The Netherlands, v. 40, p. 235-282, 1989.

[64] SATTLER, K.; SCHALLEHN, E.; GEIST, I. Autonomous Query Driven Index Tuning. *In: International Database Engineering and Applications Symposium (IDEAS)*, Washington, DC, USA, p. 439-448, 2004.

[65] LEE, S., MOON, B., PARK, C. KIM, J., Kim. S. A Case for Flash Memory SSD in Enterprise Database Applications. In SIGMOD, p. 1075-1086, 2008.

[66] SALLES, M. V. Criação autônoma de banco de dados. 2004. Dissertação (Mestrado em Informática) – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2004.

[67] MEYER, T. P.; PACKARD, N. H. Local Forecasting of High-Dimensional Chaotic Dynamics. In: CASDAGLI, N.; EUBANK, S. (Eds.), Nonlinear Modeling and Forecasting. Addison-Wesley. 1992.

[68] SHASHA, D., BONNET, P. Database Tuning – Principles, Experiments, and Troubleshooting Techniques. Morgan Kaufmann Publishers, Elservier Science, 2003.

[69] MICHALSKI, R. S.; CARBONELL, J. ; MITCHELL, T. M, Escaping brittleness: The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. In: Machine Learning. 1. ed. Los Altos: Morgan Kaufmann, 1986.

[70] MITCHELL, T. Machine Learning. New York: McGraw Hill, 1997.

[71] MORELLI, E. M. T. Recriação Automática de Índices em um SGBD Relacional. 2006. 120 f. Dissertação (Mestrado em Informática) – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUC-RIO), Rio de Janeiro, 2006.

[72] BOOKER, L. B. Improving the Performance of Genetic Algorithms in Classifier Systems. In*: International Conference on Genetic Algorithms and their Applications*, Pittsburgh, PA, USA. p. 80-92, 1995.

[73] RICHARDS, R. A. Zeroth-Order Shape Optimization Utilizing Learning Classifier Systems. 1995. 196 f. Dissertation (PhD) - Stanford University, Stanford, 1995.

[74] NA, G.; LEE, S.; MOON, B. Dynamic In-Page Logging for B+-tree Index. IEEE Transactions on Knowledge and Data Engineering, v. 24, n. 7, p. 1231-1243, 2012.

[75] FRANK, M.; OMIECINSKI, E.; NAVATHE, S. Adaptive and automated index selection in RDBMS. In: *International Conference on Extending Database Technology (EDBT*), Viena, Austria, p. 277-292, 1992.

[76] Boncz,P,A. Neumann,T. Erling, O.TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark. TPCTC, 2013: 61-76.

[77] PAPADOMANOLAKIS, S.; AILAMAKI, A. Autopart: Auotomating Schema Design for Large Scientific Databases Using Data Partitioning. In: *International Conference on Scientific and Statistical Database Management (SSDBM)*, Santorini Island, Greece. p. 357-360, 2004.

[78] CHAUDHURI, S.; NARASAYYA, V. Self-Tuning Database Systems: A Decade of Progress. In: *International Conference on Very Large Databases (VLDB)*, Vienna, Austria. p. 3-14, 2007.

[79] BAXTER, A; Storage Review. <http://www.storagereview.com/ssd_vs_hdd>

[80] Deep Learning (m)eats Databases. VLDB 2017. Keynote. <http://bit.ly/2Eb5FeW>

[81] PostgreSQL 10.1-Ext-SSD (2018) <https://github.com/wendelpedrozo/postgreSQL-10.1-Ext-SSD>

[82] TPC-H (2019) < http://www.tpc.org/tpch>.

[83] CANTÚ-PAZ, E., A Summary of Research on Parallel Genetic Algorithms. IlliGal Report No. 95007, Jul. 1995.

[84] DAS,S.,ABRAHAM, A., CHAKRABORTY, U. K., KONAR, A. Differential Evolution using a neighborhood-based mutation operator, in: *IEEE Transactions on Evolutionary Computation*, volume 13, p. 526,553, 2009.

[85] LINDEN, R. Algoritmos Genéticos. [S.l.]: Brasport, 2008. 428 p.

[86] SUTTON, R. S.; BARTO, A.G. Reinforcement Learning: An Introduction, The MIT Press, Cambridge, MA, 1998.

APÊNDICE A – REVISÃO SISTEMÁTICA DA LITERATURA

Processamento de consultas, page layout e indexação em memória Flash: uma revisão sistemática da literatura

Wendel Góes Pedrozo[1], Júlio Cesar Nievola[1], Deborah Carvalho Ribeiro[2] e Leonardo Henrique Pereira[1]

[1] Programa de Pós-Graduação em Informática - PPGIa
Pontifícia Universidade Católica do Paraná (PUC-PR), Curitiba, Brasil

[2] Programa de Pós-Graduação em Tecnologia em Saúde - PPGTS
Pontifícia Universidade Católica do Paraná (PUC-PR), Curitiba, Brasil

Resumo

O uso do armazenamento em memória flash tem proporcionado significativa melhoria no desempenho de SGBDs. No entanto, devido a maioria dos SGBDs não reconhecerem as especificidades da memória flash em seu projeto e implementação, faz com que o desempenho obtido seja abaixo do esperado. Buscando atender a esse problema, verifica-se na literatura da última década várias pesquisas com foco na memória flash, sugerindo adaptações e novas técnicas nas áreas de processamento de consulta, *page layout* e indexação. Este artigo realiza uma revisão sistemática com objetivo de identificar nos estudos como as especificidades da memória flash impactam no desempenho dos SGBDs e quais técnicas estão sendo desenvolvidas, levando em conta a especificidade da memória flash. Esta revisão sistemática identificou 1974 artigos. Após a eliminação dos títulos duplicados e não relacionados à revisão, restaram 1686 artigos. Aplicando os critérios de exclusão, o número de artigos foi reduzido para 25, sendo então devidamente analisados e classificados. Como resultado, observou-se que as características únicas da memória flash trouxeram um novo cenário para os SGBDs, sendo que as características mais exploradas nos estudos foram: *no-overwrite*, leitura e escrita assimétricas e paralelismo. Verificou-se também que novas técnicas estão sendo criadas e/ou adaptadas, sendo classificadas em técnicas para: novos modelos de custos *flash-aware*, algoritmos de junções, paralelismo, alteração de *page layout* e indexação. Verifica-se nos estudos que a utilização do armazenamento flash em SGBDs, aliado as novas técnicas desenvolvidas, tem permitido alcançar significativo incremento em desempenho no processamento de dados se comparado aos HDDs.

Palavras-chaves: memória flash, índices, *page layout*, processamento de consultas;

1. Introdução

Nos últimos anos tem ocorrido um crescimento rápido no uso da memória flash como meio de armazenamento em computadores e dispositivos móveis. Isso se deve principalmente aos seus benefícios sobre a tecnologia concorrente, o *Hard Disk Drive* (HDD), incluindo baixo consumo de energia, menor latência, menor peso e portabilidade.

Por ser um dispositivo puramente eletrônico, a memória flash possui algumas características exclusivas que as diferenciam dos HDD. Essas especificidades não são reconhecidas pela maioria dos sistemas gerenciadores de banco de dados (SGBDs), pois eles têm seus projetos e implementações internas baseados nas características dos HDD. Importantes módulos internos do SGBD, como o módulo de processamento de consultas, responsável pelo planejamento e recuperação dos dados para cada solicitação recebida, desconsideram as características únicas da memória flash, o que pode levar a um desempenho subótimo [21].

Este cenário tem impulsionado pesquisadores e fornecedores de SGBDs a repensarem o projeto interno dos SGBDs,

principalmente os métodos de acesso a dados, incluindo métodos indexados e também os modelos de estimativa de custos, sob a perspectiva da memória flash. Na literatura verifica-se que na última década um razoável esforço foi despendido por pesquisadores em busca de adaptação e de novas técnicas na área de processamento de consulta e indexação, visando o incremento de desempenho de aplicações de banco de dados. Incremento de desempenho pode ser entendido que a aplicação de banco de dados terá maior vazão (*throughput*) em termos das transações que executa ou que algumas de suas transações terão menores tempos de resposta.

Neste sentido, este artigo apresenta uma revisão sistemática da literatura (RSL), visando reunir a maior quantidade de material relevante sobre SGBDs que utilizam armazenamento em memória flash, nas áreas específicas de processamento de consultas e indexação. Mais especificamente, as questões para esta pesquisa são:

(1) Como as características únicas da memória flash impactam no desempenho de SGBDs?

(2) Quais técnicas na área de processamento de consulta e indexação estão sendo desenvolvidas levando em conta a especificidade da memória flash?

As outras partes deste artigo estão organizadas da seguinte maneira. A seção 2 define os conceitos principais relacionados aos tópicos abordados. A seção 3 descreve as principais etapas do método utilizado para elaboração da RSL. A seção 4 apresenta os resultados da revisão. A seção 5 apresenta às respostas as questões de pesquisa e conclusão.

2. Referencial Teórico

A revisão sistemática da literatura (RSL) é uma revisão metodologicamente rigorosa dos resultados da pesquisa sobre determinado objeto. Uma RSL é um meio de identificar, avaliar e interpretar toda a pesquisa disponível e relevante para uma questão de pesquisa específica, área de tópico ou fenômeno de interesse [24, 25].

Para melhor compreensão do tema e das questões de pesquisa deste trabalho, na seção 2.1 serão apresentadas as especificidades técnicas da memória flash e suas principais diferenças em relação ao HDD. Na seção 2.2 são introduzidos conceitos de processamento de consultas e indexação, que é onde esse trabalho se situa dentro da grande área de banco de dados.

2.1 NAND flash

A memória NAND flash é à base de construção de um *Solid-State Drive* (SSD). Um chip flash compõe-se de uma cadeia de células flash, que podem armazenar bits 0 ou 1.

Conforme ilustrado na figura 1, na estrutura hierárquica de um SSD, um pacote de memória flash é composto por vários chips. Um chip apresenta vários planos, e cada plano contém um conjunto de blocos, sendo estes compostos de 64 a 256 páginas cada um [29].

São três as ações que podem ser executadas em um dispositivo flash: ler, programar/escrever e apagar.

O estado inicial de cada célula de memória flash é definido como "1" e pode ser alterado para "0" por operação de programação (ou escrita). Uma vez que a célula é programada para "0", a reversão para o estado "1" original requer a operação de apagar *(erase)*, que somente poderá ser executada em um bloco inteiro.

A seguir, tem-se um resumo das características únicas da memória flash, que a diferencia de outros dispositivos de armazenamento:

- *No-overwrite*: em memória flash uma página é a unidade básica de operações de leitura e escrita, enquanto as operações de *erase* são executadas por blocos. Portanto, ao contrário do que acontece em um HDD, na memória flash não é possível alterar seletivamente uma página específica em um bloco. Para substituir os dados existentes, primeiro um bloco inteiro deve ser limpo *(erase)* e somente após isso os novos dados podem ser escritos. O processo é conhecido como *erase-before-write*, o que significa que todas as outras páginas do bloco que não necessitam de alteração, precisam ser lidas também, carregadas para a memória principal e, em seguida, escritas novamente [19];
- Ágeis leituras aleatórias: *a m*emória flash é um dispositivo puramente eletrônico, sem partes mecânicas móveis. Desta forma, não existe latência de busca e/ou rotação como nos HDDs, permitindo assim que leituras aleatórias tenham desempenho similar a leituras sequenciais. Além disso, como pode ser visto na tabela 1, operações de leitura em memória flash podem ser mais de 300 vezes mais rápidas que leituras em HDD [11].
- Leitura e escrita assimétricas: memória flash tem velocidade assimétrica de leitura e escrita. Como reportado na

tabela 1, operações de leitura podem ser mais de 8 vezes mais rápidas que a escrita. Isso ocorre porque gasta-se mais tempo para energizar uma célula do que simplesmente ler o seu estado.

- Limitado número de escrita: normalmente variando entre 10.000 a 100.000 ciclos por bloco. Após exceder esse limite, o bloco torna-se não confiável. Para evitar que alguns blocos se tornem inutilizáveis muito antes de outros no mesmo dispositivo, são empregadas técnicas que distribuem o nível de desgaste uniformemente em todo segmento de memória [8];

- Paralelismo: pode ser obtido de quatro formas: (i) em nível de canal, podendo ser operados independente e simultaneamente; (ii) em nível de pacotes flash ligados ao mesmo canal, permitindo que um canal seja compartilhado por vários pacotes simultaneamente (figura 1); (iii) o próximo nível é o acesso simultâneo a chips dentro de um pacote; e (iv) paralelismo em nível de plano, permitindo que as operações (leitura, escrita ou *erase*) possam ser executada simultaneamente em vários planos [46].

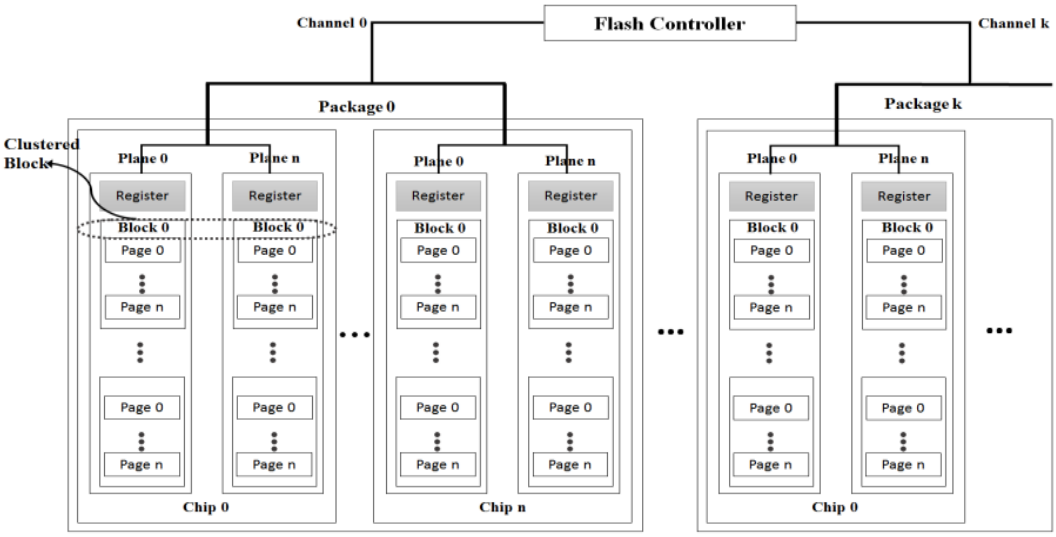


Figura 1. Arquitetura interna de um Flash SSD.

Tabela 1 – Medidas de tempo de acesso para HDD e SSD

	Tempo de acesso		
	Leitura	Escrita	Apagar (*erase)*
HDD	20,4 ms (4 KB)	21,5 ms	-
NAND Flash	0,06 ms (4 KB)	0,5 ms (4 KB)	1.5 ms (256 KB)

As características singulares da memória flash, como *no-overwrite* e o limitado número de escritas, requerem algumas ações adicionais. Nos SSDs atuais, existe um componente de hardware chamado *Flash Translation Layer* (FTL) responsável por essas ações, a saber: (i) nivelamento de desgaste, responsável por distribuir operações de escrita para os blocos menos utilizados do disco; (ii) coleta de lixo, processo assíncrono para apagar os blocos antigos para reutilização futura e (iii) mapeamento e tradução de endereços.

2.2 Processamento de consultas e indexação

Entre as diversas áreas de banco de dados, neste trabalho, como mencionado na seção 1, serão abordados estudos na área de processamento de consulta e indexação desenvolvidos sob a perspectiva da memória flash (*flash-aware*).

Entre as várias funções do módulo de processamento e otimização de consultas, uma das principais é a elaboração de um plano de execução da consulta, que consiste em escolher quais os métodos de acesso, serão utilizados pelo subsistema de armazenamento. Em otimizadores de consultas baseados em custo, para escolha do melhor plano de execução é analisado entre vários planos possíveis, o plano com menor custo. Esse custo é obtido pelas estatísticas coletadas e pelo modelo de custos de cada SGBD. As estatísticas permitem obter dados e informações aproximadas sobre relações, como

a seletividade e a cardinalidade. Já o modelo de custo possui vários fatores sobre os recursos computacionais necessários de cada método de acesso, como: a quantia de memória principal necessária, a quantidade de acessos sequenciais/aleatórios em armazenamento secundário e o tempo de utilização do processador.

Concernente à indexação, vale lembrar que, embora opcional, o uso de índices pode melhorar significativamente o desempenho de consultas em banco de dados. Índices correspondem a uma estrutura lógica e ordenada que mapeia os endereços dos dados armazenados fisicamente [47]. Métodos de acesso baseados em índices normalmente têm ganho de desempenho em operações de ordenação, agrupamento e classificação de dados [47], sobretudo no caso de uso de armazenamento em flash, devido às rápidas operações de leitura em acesso aleatório. No entanto, requisições de *insert*, *delete* ou *update*, podem ocasionar uma sobrecarga em operações de escrita, devido à restrição de *erase-before-write* da memória flash.

3. Método

Uma revisão sistemática da literatura envolve várias atividades distintas, as quais, de acordo com [49], se resumem em três fases principais: planejar, conduzir e relatar a revisão. De forma geral, uma RSL consiste em especificar a questão de pesquisa, definir a estratégia de busca, documentar a estratégia de busca, realizar a avaliação e estabelecer conclusões [48, 49].

As próximas subseções detalham as etapas executadas para produzir este estudo.

3.1 Planejando a revisão

O planejamento começou com o desenvolvimento de um protocolo para a revisão sistemática, especificando o processo e os métodos aplicados. Como ilustrado na figura 2, as etapas do protocolo incluem essencialmente as questões de pesquisa (definidas na seção 1), a estratégia de busca, os critérios de seleção de estudos primários, e a síntese dos dados.

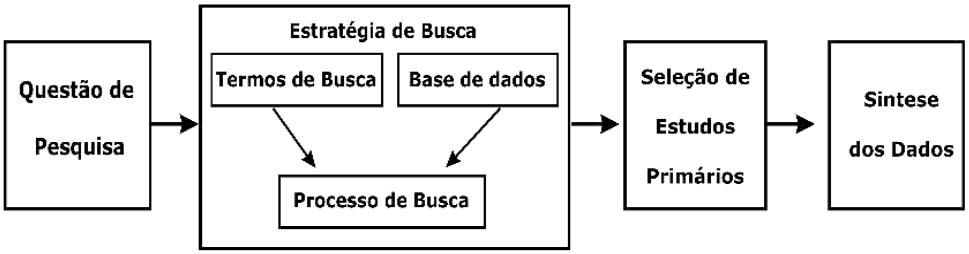


Figura 2. Estágios do protocolo da revisão

3.2 Conduzindo a revisão

Uma vez que o protocolo tenha sido acordado, a revisão pode começar [49]. Assim, as etapas desta fase são descritas a seguir.

3.2.1 Estratégia de Busca

Esta fase compreende três importantes subetapas dependentes entre si (figura 2), consistindo em: definir os termos de busca; definir as fontes da literatura, ou seja, as bases de dados; e, por fim, realizar o processo de busca.

3.2.1.1 Termos de busca

Um dos objetivos de uma RSL é encontrar o maior número de estudos primários possíveis relacionados às questões de pesquisa, utilizando uma estratégia de busca imparcial [27]. Para isto, o primeiro passo é identificar as palavras-chave e termos de pesquisa a serem pesquisados em todas as bases de dados utilizadas.

Para encontrar as palavras-chave, os seguintes critérios foram utilizados:

a) derivar maior número de termos relacionados à questão de pesquisa;
a) identificar grafias alternativas e sinônimos para grandes termos;
b) verificar as palavras-chave em artigos e livros relevantes;
c) adotar o uso do OR Lógico para incluir grafias alternativas e sinônimos;

d) usar o AND lógico para ligação dos principais termos.

Como resultado desta etapa, os termos de busca elencados foram:

(*"Flash memory" OR "Flash Disk" OR "Solid State Drives" OR "SSD" OR "SSS" OR "Solid State Disk" OR "Solid State Storage"*) *AND* (*"Database" OR "DBMS" OR "RDBMS" OR "ORDBMS"*) *AND* (*"Index" OR "Indexing"*) *AND* (*"query processing" OR "query optimization" OR "query optimizer"*).

3.2.1.2 Bases de dados

Para que a pesquisa dos estudos primários fosse realizada, foram definidas as fontes da literatura, por meio dos termos de busca definidos.

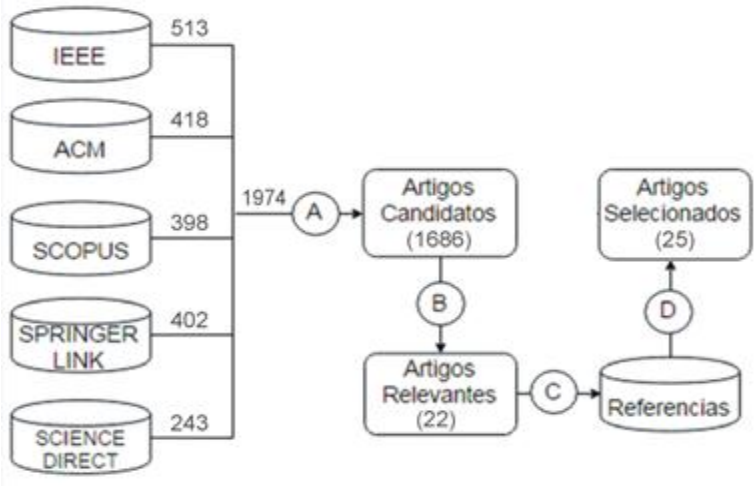
Desta forma, foram analisadas várias bases de dados candidatas e, finalmente, escolhidas as cinco bases de dados digitais a seguir: *IEEE Xplore, ACM Digital Library, Springer Link, Science Direct e Scopus.* Outras bases de dados não foram consideradas, uma vez que são praticamente cobertas pelas bases selecionadas.

Os termos de busca necessitaram de pequenos ajustes nas diferentes bases de dados, uma vez que os motores de busca utilizam sintaxes diferentes de pesquisa. A pesquisa com os termos de busca foi realizada em "texto completo" nas bases de dados: IEEE Xplore, ACM Digital Library, Springer Link e Science Direct. Na base de dados Scopus foi pesquisado somente em título, resumo e palavras-chave, pois a mesma não permite pesquisa em texto completo.

A pesquisa foi restringida ao período de 1º de janeiro de 2000 a 31 de dezembro de 2018. Destaca-se que o início foi somente a partir do ano 2000, pois, embora os dispositivos baseados em memória flash tivessem surgido na década de 1980, somente a partir do ano 2000 é que estes foram lançados no mercado com capacidades na faixa de terabytes, possibilitando assim o seu uso comercial em SGBDs.

3.2.1.3 Processo de busca

Uma RSL requer uma pesquisa abrangente. Com os termos de pesquisa e as bases de dados definidos anteriormente, o processo de busca pôde ser iniciado. Como ilustrado na Figura 3, foi realizada uma busca individual em cada uma das cinco bases de dados digitais, obtendo 1974 artigos ao todo. Depois foi feita a remoção de artigos duplicados (rótulo A), reduzindo para 1686 os artigos candidatos. Na sequência, foram aplicados critérios de seleção (rótulo B), obtendo 22 artigos relevantes. Seguindo o processo, foram realizadas pesquisas nas listas de referência dos artigos relevantes obtidos (rótulo C), visando encontrar estudos adicionais relevantes num processo conhecido como *snowbaling*. Após serem feitas buscas adicionais nas listas de referências (rótulo D), foram encontrados três artigos extras, que não estavam no rol de estudos candidatos, mas que atendiam aos critérios de seleção. A Figura 3 apresenta uma visão geral e os resultados obtidos em cada etapa. A lista completa dos estudos selecionados pode ser vista na Tabela 2.



Etapa A – Remover duplicados
Etapa B – Aplicar critérios de seleção
Etapa C – Verificar referências de artigos relevantes
Etapa D – Busca adicional por artigos relevantes

Figura 3. Busca e processo de seleção

3.2.2 Seleção de estudos primários

Na seção anterior, foi apresentada uma visão geral do processo de busca. Na presente seção, são mostradas detalhadamente as fases internas para aplicação dos critérios de seleção (Figura 3 - rótulo B).

Resultantes do processo de busca da seção anterior, foram reunidos 1686 artigos candidatos (ver Figura 3). Uma vez que muitos dos artigos candidatos não forneciam informações úteis relacionadas às questões de pesquisa levantadas por esta revisão, foi necessário fazer uma filtragem adicional para identificar os artigos relevantes. O processo de seleção dos estudos consistiu em aplicar critérios de inclusão e exclusão (definidos no parágrafo a seguir) aos artigos candidatos, de modo a identificar os trabalhos relevantes, que fornecem dados potenciais para responder às questões de pesquisa.

Os critérios que definem a inclusão de um artigo implicam que o estudo aborde o uso de armazenamento em memória flash em SGBDRs, visando propor soluções na área de indexação ou processamento de consultas. Para exclusão de resultados os seguintes critérios foram definidos: artigos que abordem SGBDs não relacionais; e artigos que são apenas revisão da literatura, *surveys* ou similares.

3.2.3 Síntese dos dados

O objetivo da síntese de dados é agregar evidência dos estudos selecionados a fim de responder às questões de pesquisa. Na síntese identificamos os principais conceitos de cada estudo primário. Esses conceitos principais foram então organizados em forma de tabela para permitir comparações entre estudos, visão dos resultados encontrados em alto nível e classificação, em áreas. Em suma, a síntese foi obtida por: (i) identificar um conjunto de trabalhos com uma mesma classificação relativa; (ii) documentar características principais de cada estudo; e (iii) mensurar aspectos qualitativos e quantitativos do conjunto de estudos obtido.

4. Resultados

Esta seção apresenta e discute os resultados desta revisão. Primeiro, apresentamos uma visão geral dos estudos selecionados. Em seguida, relatamos e discutimos os resultados encontrados segundo as questões de pesquisa definidas (seção 1.1), discutindo cada uma delas em subseções separadas.

4.1. Visão geral dos estudos

4.1.1 Ano da publicação

A Figura 3 apresenta o número de estudos realizados ao longo dos anos. Vale lembrar que, embora a pesquisa tenha sido a partir do ano 2000, em alguns períodos/anos não ocorreram publicações, como, no período de 2000 a 2006, por exemplo. Como pode ser visto, o pico de publicações se deu de 2010 a 2011 com cinco estudos para cada ano e um segundo ciclo de interesse foi entre 2013 e 2014 com 3 estudos.

4.1.2. Local de publicação

Dos 25 estudos, quase metade foi publicada em *journals* e o restante em anais de conferências. A quantidade de publicações por locais é exibida na Tabela 2. Os locais de publicação incluem, sobretudo, os periódicos *IEEE Transactions On Computers*, *The VLDB Journal* e *SIGMOD Record*. Estes são periódicos consolidados e com alto fator de impacto na área de sistemas de banco de dados, sendo que 30% dos estudos originam deles.

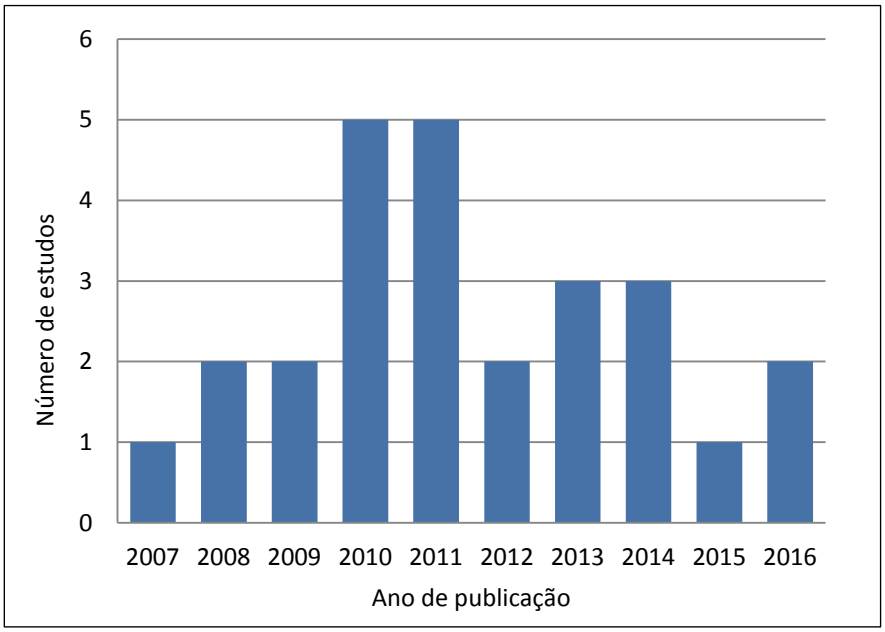


Figura 4. Publicações ao longo dos anos

Tabela 2. Locais de publicação e distribuição dos estudos selecionados

Local da Publicação	**Sigla**	**Tipo**	**Nº de artigos**	**Percentagem**
International Database Engineering & Applications Symposium	IDEAS	Conferência	1	7.4
The VLDB Journal	VLDB	Jornal	4	14.8
ACM Transactions on Embedded Computing Systems	TECS	Jornal	1	3.7
SIGMOD Record	SIGMOD	Jornal	1	3.7
Data Management on New Hardware	DaMoN	Conferência	2	7.4
SIGMOD International Conference on Management of data	SIGMOD	Conferência	2	7.4
International Conference on System Science, Engineering Design and Manufacturing Informatization	ICESEM	Conferência	1	3.7
IEEE Transactions On Computers		Jornal	3	11.1
Web Information System and Application Conference	WISA	Conferência	1	3.7
Journal of Computer Science and Technology		Jornal	1	3.7
Information Processing in Sensor Networks	IPSN	Conferência	1	3.7
ACM Symposium on Applied Computing	SAC	Conferência	2	7.4
IEEE Transactions on Knowledge and Data Engineering		Jornal	1	3.7
Lecture Notes in Computer Science	LNCS	Jornal	1	3.7
International Conference on Information and Automation	ICIA	Conferência	1	3.7
The Journal of Systems and Software		Jornal	1	3.7
Information Sciences	IS	Jornal	1	3.7
Total			25	100

4.1.3 Testes comparativos utilizados nos estudos selecionados

Um importante fator em artigos científicos é que este tenha, em seu corpo, algum processo de comparação da solução proposta com outras similares, e que seja conciso e imparcial. A tabela 3 apresenta informações de como foram os testes comparativos de cada estudo selecionado. Além de informações básicas como autor e nome da solução/algoritmo, foram extraídos também o nome do trabalho que serviu de comparação em testes de desempenho ou análise. Outra informação importante extraída, é sobre o conjunto de dados de testes utilizado para medir o desempenho da solução, ou seja, se foi utilizado algum *benchmark* para geração de dados ou se foi utilizado alguma alternativa particular para isto.

Analisando os dados constantes na tabela 3, pode-se verificar que o trabalho denominado BFTL [2] é o mais utilizado como base de comparação para outros trabalhos (6 comparações). Isto se deve principalmente ao fato de ser o trabalho selecionado mais antigo no contexto desta RSL e um dos precursores na área de indexação aplicada a armazenamento flash. Outro ponto relevante é que dos 25 estudos selecionados 17 implementaram testes comparativos práticos com outras soluções similares, destacando a qualidade dos estudos selecionados e evidenciando as características e avanços de um trabalho em comparação com outros.

Sobre o conjunto de dados de teste e carga de trabalho utilizados, foi possível verificar que os *benchmarks* TPC-H e TPC-C foram utilizados mais vezes, perfazendo 9 casos. Outros *benchmarks*/*tools* correspondem a 5 trabalhos e o restante (11 trabalhos) utilizaram conjuntos particulares de dados.

Tabela 3 – Testes comparativos utilizados nos estudos selecionados.

1º Autor /Ano	Ref.	Nome da Solução/Algoritmo	Teste comparativo com outra solução	Dataset/ Benchmark
Lee (2011)	[1]	Index-Based Partition Sort	N/A	Próprio
Wu (2007)	[2]	BFTL	FTL (padrão)	Próprio
Roh (2012)	[3]	PIO-B+Tree	BFTL, FD-Tree	TPC-C
Agrawal (2009)	[4]	LA-Tree	BFTL, FlashDB, IPL	TPC-C
Li (2010)	[5]	FD-Tree	B+Tree, LSM-Tree e BFTL	Próprio
Nath (2007)	[6]	FlashDB	N/A	LabData
Jiang (2014)	[7]	AB-Tree	B+Tree, BFTL e FD-Tree	Próprio
Fang (2014)	[8]	AD-Tree	B+Tree e LA-Tree	Próprio
On (2010)	[9]	Lazy Update B+Tree	B+Tree	DBLP
Ahn (2013)	[10]	U*-Tree	BFTL	TPC-C, Kerner, Postmark, financial, genereal Web
Wang (2010)	[11]	Flash-based extendible hash index	Hash Index	Próprio
Byun (2008)	[12]	CHC-Tree	B+Tree	Próprio
Evangelista (2015)	[15]	BT-Join (B+BT-Tree)	Flash-Join	TPC-H
Shah (2008)	[16]	RARE-Join e GRACE-PAX	GRACE-HASH	TPC-H
Tsirogiannis (2009)	[17]	Flash-Join	Hybrid-Hash Join	TPC-H
Li (2013)	[18]	Dijest Join	N/A	TPC-H
Nath (2007)	[19]	d-IPL B+Tree	IPL B+Tree e BFTL	Próprio
Lai (2013)	[20]	Para-Scan, Para-Hash-Join	Hash-Join	TPC-H
Bausch (2012)	[21]	Novo modelo custo	N/A	TPC-H
Ghodsnia (2014)	[22]	QDT Model	DTT-Model	Próprio
Ash (2014)	[23]	Two-level minipage	N/A	I/O Meter tool

4.2 Impacto da memória flash no desempenho de SGBDs (QP1)

Considerando a primeira questão de pesquisa (QP1), verificou-se que as características únicas da memória flash trouxeram um novo cenário para os SGBDs, sendo que o impacto no desempenho, conforme a bibliografia analisada, tem sido positivo, salvo algumas exceções, para operações específicas.

Na figura 4 são listadas as características únicas da memória flash e o número de trabalhos primários que as contemplaram como objeto de pesquisa.

É possível perceber que a característica de *no-overwrite* é a característica mais explorada nos estudos selecionados. *No-overwrite* é considerado um dos gargalos na utilização de armazenamento flash em SGBDs. Sendo abordado principalmente por trabalhos de indexação, visando minimizar o problema de sobrecarga de escrita, pois, normalmente a inserção de um novo elemento em um índice de arvore B+, pode ocasionar a atualização de dados e metadados em vários outros nós, para satisfazer o requisito de balanceamento da arvore B+, que por sua vez causa uma série de operações de escritas aleatórias (*random write*) [42].

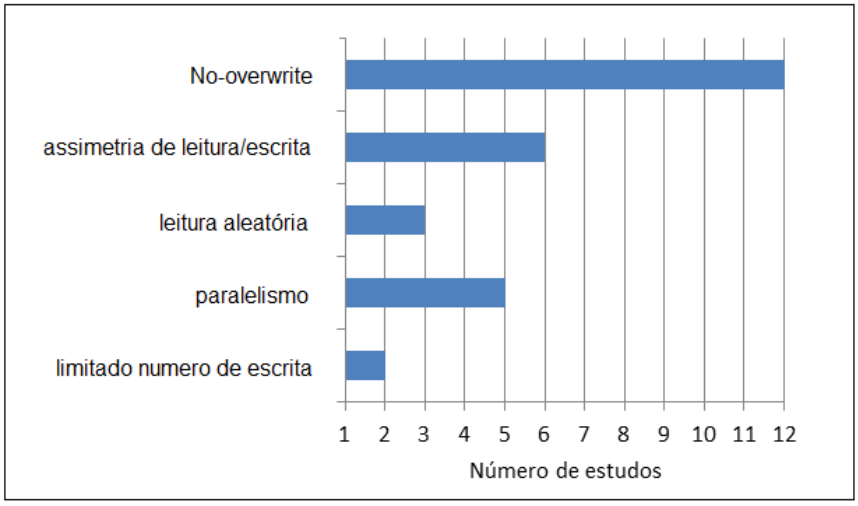


Figura 4 – características principais exploradas em pesquisas sobre armazenamento em memória *flash*

Outra característica da memória flash, com significativo impacto na melhoria do desempenho, contando com seis estudos [11, 14, 16, 40], é a velocidade assimétrica entre leitura e escrita. Além disso, destaca-se também a velocidade de leitura aleatória, que devido à ausência de latência mecânica, é bem próxima da velocidade de acesso sequencial [40]. Considerando que em memória flash, uma operação de leitura pode ser mais de 300 vezes mais rápida que em um HDD, essa característica demonstra vantagens para o incremento do desempenho.

A possibilidade de utilizar paralelismo em operações internas da memória flash também é um assunto explorado, sendo objeto de cinco estudos. Apesar de representar uma vantagem em comparação com o HDD, o paralelismo ainda não é completamente utilizado por muitos SGBDs, pois como demonstrado em [1, 20], para usufruir dos vários níveis de paralelismo, é necessário que algoritmos internos do SGBD como *hash join*, *scan* e *index scan*, sejam alterados.

4.3 Novas técnicas *Flash-aware* desenvolvidas (QP2)

Concernente a QP2, verificou-se que iniciativas e novas técnicas para processamento de consultas, *page layout* e indexação estão em constante desenvolvimento, visando atender à especificidade da memória flash. Na tabela 4 é apresentada uma classificação das técnicas identificadas, sendo que cada uma delas é comentada e discutida nas subseções a seguir.

Tabela 4 – Classificação das técnicas identificadas

Categoria	Subcategoria	Artigo científico
Processamento de consultas	Modelo de custos *Flash-aware*	[21, 22]
	Algoritmos de Junções	[15,16,17,18,20,36]
	Paralelismo	[1,20]
Page layout	*Page layout*	[15, 16, 17, 23]
Indexação	*FTL-Based*	[2,3,5,7,9,12,23,37,43]
	No FTL-Based	[4,8,10,19]
	Hashing	[11, 23]

Devido à natureza específica das diferentes técnicas e das características de cada estudo analisado, na tabela 5 e 6 são listadas as características gerais de cada estudo selecionado.

4.3.1 Modelo de custos *Flash-aware*

Atualmente, os mais populares SGBDs têm seus modelos de custos baseados em propriedades dos HDDs, não considerando as características e especificidades da memória flash. Neste sentido, uma das principais ações do otimizador – que é a escolha dos métodos de acessos para uma consulta, pode ter um resultado subótimo.

Estudos como de [21] e [22] propõem novos modelos de custos que consideram as características da memória flash e

também dos HDDs, além de explorar e usufruir do paralelismo da memória flash. Na tabela 5 são listados esses estudos, incluindo uma síntese das características de cada estudo, o nome dado a solução e seu número de referência.

Tabela 5 – síntese dos estudos selecionados para a área de processamento de consulta e *page layout*.

Ref.	Nome	Descrição
[1]	SIDX	Lee et al. (2011) propõem um algoritmo de classificação externa baseado em índice denominado SIDX (Sorted Index Scan), que explora as informações da distribuição de valores-chave nos nós folhas de índices *non-clustered*, visando a utilização do paralelismo interno da memória flash.
[15]	Bt-Join	Evangelista et al. (2014) propõem o algoritmo Bt-Join, que visa reduzir a quantidade de operações de escrita durante a execução de junções. O algoritmo usa uma versão estendida da árvore B+ a fim de computar eficientemente uma operação de junção, conforme o tipo de layout de página utilizada no subsistema de armazenamento (*collumn-based* ou *row-based*).
[16]	RARE-join	Shah et al. (2008) propõem o algoritmo RARE-join, que visa o aproveitamento das rápidas leituras aleatórias da memória flash. Assim, os algoritmos de E/S tradicionais são convertidos em algoritmos que usam uma mistura de E/S sequencial e aleatória, a fim de processar menos dados.
[17]	FlashJoin	Tsirogiannis et al. demonstra como o layout de armazenamento PAX reduz a quantidade de dados lidos durante seleções e projeções. É proposto o algoritmo de junção em pipeline denominado FlashJoin, que reduz significativamente o acesso a memória, por acessar apenas os atributos de junção, produzindo resultados parciais na forma de um índice de junção.
[18]	DigestJoin	Li et al. (2012) propõem um framework chamado DigestJoin, que tem por objetivo reduzir resultados intermediários além de explorar as rápidas leituras aleatórias da memória flash.
[20]	ParaHashJoin / ParaScan	Em Lai et al. (2013), o paralelismo é explorado através do uso de múltiplas threads, com vistas a otimizar operações de varredura e junção através de dois novos algoritmos, a saber: (i) ParaScan, direcionado a operações de varredura; e (ii) ParaHashJoin direcionado a operações de junção.
[21]	Novo modelo custo	Bausch et al. (2012) propõem um novo modelo de custos que considera a assimetria de leitura/escrita da memória flash. São criados novos parâmetros de custos que diferenciam custos de leitura sequencial, escrita sequencial, leitura aleatória e escrita aleatória. Além disso, foram adaptados alguns métodos de acesso (*external sort, hash join, sequential scan* e *index scan*) para funcionarem com os novos parâmetros criados.
[22]	QDTT Model	Ghodsnia et al. (2014) propõem um novo modelo de custos chamado *Queue-depth-aware disk transfer time* (QDTT), que diferencia custos de E/S de diversos dispositivos de armazenamento, além de considerar os custos de E/S utilizando o paralelismo. Foram propostos dois novos algoritmos para funções de varredura – em tabelas e em índices – visando ativar o uso do paralelismo interno da memória flash.
[23]	Two-level minipage	Ash et al. (2014) investigaram como escolher o *page-size* que melhor otimize o *trade-off* de pequenas e grandes requisições de E/S para operações de varredura. Foi proposto um novo esquema chamado *Two-level minipage layout* para aumentar os tradicionais *slotted page* (NSM) de discos SSDs.
[38]	N-Hash	Shin et al. (2016) propõem um novo algoritmo de particionamento de *hash*, chamado (N-Hash), visando otimizar a materialização e recuperação de resultados intermediários.

4.3.2 *Page layout*

Um dos problemas em dispositivos flash é como lidar eficientemente com pequenos *random writes*, decorrentes da atualização de dados e metadados, causado pelo processo de *erase-before-write*. *Randon writes* causam muitas operações de *erase*, que por sua vez desencadeia operações de leitura e escrita em disco.

Alteração de layout de página é uma técnica que pode prevenir, entre outros problemas, pequenos *random writes*, além de maximizar as operações de leitura. Uma abordagem para mudança de layout de página denominada PAX *(Partition Attribute Across),* também conhecida como modelo *column-based* de armazenamento, divide cada relação de forma horizontal e vertical. Em suma, para uma relação de n-atributos, tuplas são alocadas para páginas PAX, que cria minipáginas dentro de cada página, onde cada minipágina armazena dados para um atributo (coluna) da relação. Assim, durante o processamento de consultas que exigem apenas alguns atributos das tuplas, a quantidade de operações de leitura é reduzida, pois apenas um subconjunto de minipáginas é acessado, melhorando assim o desempenho de E/S.

Embora o modelo PAX seja particularmente otimizado para ambientes de leitura intensiva, como aplicações OLAP *(Online Analytical Processing),* ao aplicar PAX em ambientes OLTP (On-line Transaction Processing) este sofre com sobrecargas de escritas – conhecida fraqueza do modelo *column-based* de armazenamento.

Tabela 6 – síntese dos estudos selecionados para a área de indexação.

Ref	Nome	Descrição
[2]	BFTL	Wu et al. (2007) propõem o BFTL, o primeiro índice *flash-aware* projetado sobre a FTL, que tem por objetivo minimizar a quantidade de escritas geradas pela limitação de *no-overwrite* da memória flash. BFTL grava em log as operações de *insert/update/delete*, e transfere para a memória flash quando a dimensão do *slot buffer* encher.
[3]	PIO-B-Tree	Roh et al. (2012) propõe PIO (Psync I/O) B-Tree, um índice variante de B+Tree que explora um novo conceito de requisição de E/S, onde o paralelismo interno da memória flash é ativado, utilizando chamadas ao sistema operacional, passando como entrada um *array* de requisições de E/S.
[4]	LA-Tree	Agrawal et al. (2009) propõem um índice baseado em B+Tree, que implementa um número fixo de sub-árvores contendo segmentos de buffer adjacentes, visando assim armazenar e executar tarefas de atualizações em lotes. Os *buffers* são posteriormente mesclados com a árvore por meio de um algoritmo adaptativo on-line, capaz de identificar *trade-off* entre desempenho de *update* e de pesquisa.
[5]	FD-Tree	Li et al. (2010) propõem um índice em árvore denominado FD-Tree, que implementa múltiplos níveis de índices em cascata. Nos níveis mais altos, existe uma pequena B+Tree chamada *head tree*. Em níveis mais baixos, os nós da B+Tree são armazenados em páginas contíguas. Operações de *updates* são aplicadas somente na *head tree* e gradualmente migrados para os níveis inferiores, em operações em lotes, visando assim minimizar o problema de sobrecarga causados por *random writes*.
[6]	FlashDB	Nath e Kansal (2007) propõem FlashDB, um banco de dados *Flash-based* que usa um novo *self-tuning* índice que adapta dinamicamente sua estrutura de armazenamento à carga de trabalho e ao dispositivo de armazenamento subjacente. Visando minimizar o problema da degradação de performance em operações de escrita, adaptando as cargas de trabalho em dois grupos *disk-type* e *log-type* [2].
[7]	AB-Tree	Jiang et al. (2014) propõem uma variante de índice B+tree denominado AB-Tree (Adaptive Batched Tree), que implementa uma estrutura *bucked-based* para mesclar pequenos *random writes*, de forma que as inserções sejam feitas sequencialmente e em lotes, melhorando assim o desempenho em operações de escrita. Para operações de *update* é adotada uma estratégia preguiçosa, minimizando assim pequenos *random writes*.
[8]	AD B+Tree	Fang et al. (2013), inspirado no trabalho de [4] propõem AD B+Tree, uma variante da B-Tree, que visa explorar o impacto de acessos a *hot-data*. O trabalho aborda o problema de propagação de *updates* e o impacto de diferentes tipos de cargas de trabalho sob uma estrutura de índice, visando a melhoria de desempenho.
[9]	Lazy-update B+Tree	On et al. (2010) propõem Lazy-update B+Tree. O método consiste na ideia de adiar comandos de *commit*, demandados por comandos de *update*, utilizando para isso um segmento de *buffer*. Após isso são realizados *commits* em grupos, obtendo a amortização do custo de cada operação e a redução das operações de escrita.
[10]	U*-Tree	Ahn et al. (2010) inspirado em [34] propõem o índice U*-Tree, com recursos de gerenciamento de flash *pages* e *garbage collection*. U*-Tree se baseia em armazenar em uma *flash-page*, o caminho das folhas até a raiz, de todos os nós atualizados. U*-Tree generaliza o *page layout*, de modo que o tamanho do nó pode ser configurado em uma granularidade fina. Desta forma é possível encontrar o melhor *page layout* que maximiza a eficiência e desempenho, além da redução de custos principalmente na reorganização de índices.
[11]	Self-adaptive extendible hash index	Wang e Wang (2010), visando minimizar o custo de E/S de atualização de índices que utilizam armazenamento flash, propõem um índice, que assim como [24] grava em log operações de escrita e utiliza estrutura *hash*. O trabalho divide um *bucket* em regiões de dados e regiões de logs. Quando uma operação de *update* ou *delete* é solicitada, o algoritmo escreve isto na região de log ao invés de alterar os dados localmente. Assim, custosas operações de *erase* são postergadas ao máximo até que o *bucket* esteja cheio.
[12]	CHC-Tree	Byun e Hur (2008) propõem CHC-Tree, uma técnica para compressão de dados pouco usados, visando redução no número de operações de escrita em um nó para comandos de *insert/delete*. CHC-Tree avalia e classifica cada nó do índice em rótulos: *hot* ou *cold*; através da informação *update-timestamp* encontrada em cada nó.
[19]	d-IPL B+Tree	Na et al. (2012) propõem uma versão estendida do algoritmo para implementação de índice IPL B+Tree, denominada d-IPL (Dynamic-In-Page Logging), que em contraste com técnicas convencionais, onde dados de log são acrescentados sequencialmente, IPL leva em consideração a limitação de *no-overwrite* da memória flash, colocando registros de log com suas correspondentes páginas de dados no mesmo flash *block*, evitando assim operações de *random writes*.
[37]	BloomTree	Jin et al. (2016) propõem um novo índice em árvore chamado BloomTree, com o objetivo de reduzir gravações e leituras. Para isso são utilizados: *buffer* de atualização, páginas de estouro e Filtros de Bloom para otimizar o desempenho de leitura e reduzir operações de *random writes*.
[43]	MB-Tree	Roh et al. (2009) propõe uma extensão de índice B-Tree chamada MB-Tree, que tem por objetivo a redução de escrita e do tempo de pesquisa. As solicitações de atualização por nó folha são acumuladas em um *buffer*, para posteriormente ser realizado o processamento simultâneo e em lote dessas solicitações.

Entre os estudos selecionados nesta RSL, verifica-se que alguns trabalhos que exploram algoritmos de junções utilizam o modelo de armazenamento *column-based* PAX, como os trabalhos de [16,17 e 36]. A explicação resumida desses e dos demais estudos podem ser retomadas na tabela 5, em que estão indicados pelo nome dado à solução e seu número de referência.

4.3.3 Algoritmos de Junção

A Junção é uma das mais custosas operações em consultas SQL, pois normalmente requer grande quantidade de acessos (leitura/escrita) em memória secundária. Para lidar com este problema e visando minimizar resultados intermediários, trabalhos de [15,16,17,18,20] utilizam a estratégia de recuperar somente as colunas necessárias para a operação de junção. De outra forma, os trabalhos de [15, 16, 17 e 36] utilizam PAX (ver seção 4.3.2) ou métodos híbridos como layout de página de armazenamento, visando a redução de leitura/escrita durante a operação de junção de relações.

Na tabela 5 pode ser encontrado o nome, número de referência e a síntese de cada um dos estudos mencionados.

4.3.4 Paralelismo

Novas técnicas em processamento de consultas se concentram principalmente em obter vantagens nas rápidas operações de leitura e acessos aleatórios do armazenamento flash. No entanto, ainda não conseguem explorar o rico paralelismo interno da memória flash. A utilização do paralelismo, exige alterações importantes no projeto interno de SGBDs, pois a maioria deles foi projetado não considerando a possibilidade de utilizar paralelismo.

Trabalhos como [20] e [22], inovam pela proposta de novos algoritmos para varreduras que consideram o uso de paralelismo. Já o trabalho de [38] utiliza técnicas de *hash* visando também a utilização do paralelismo. Na tabela 5 esses e outros estudos são listados e detalhados.

4.3.5 Índices *Flash-aware*

Ao analisar as várias técnicas de indexação *flash-aware* existentes, é importante identificar para qual tipo de aplicação é destinada a solução. Aplicações para sistemas embarcados ou dispositivos móveis normalmente utilizam *NAND Flash* com limitações de espaço de memória e ausência de uma FTL. Em contrapartida, aplicações OLTP e OLAP, utilizam SSDs que dispõem de uma FTL, permitindo que a implementação de índices seja realizada sobre a FTL.

Devido à presença da FTL ser uma característica determinante para o desenvolvimento de aplicações, os estudos selecionados foram classificados em *FTL-based*, NO-*FTL-based*, além da classificação de estudos que utilizam *hash*.

Na tabela 5, os estudos sobre indexação são listados e classificados, além de apresentados por nome, número de referência e a síntese de cada estudo.

5. Conclusão

Esta revisão sistemática teve como objetivo responder às seguintes perguntas:

(1) Como as características únicas da memória flash impactam no desempenho de SGBDs?

(2) Quais técnicas na área de processamento de consulta e indexação estão sendo desenvolvidas levando em conta a especificidade da memória flash?

Respondendo à primeira pergunta, verificou-se que as características únicas da memória flash trouxeram novos desafios a SGBDs sendo que as características mais exploradas nos estudos são: : *no-overwrite*, leitura e escrita assimétricas, paralelismo e limitado número de escritas.

No-overwrite ganhou destaque em um número considerável de estudos desta RSL. Devido à restrição de erase-before-write e consequente sobrecarga de operações de escrita, que afeta principalmente operações com índices, essa característica é considerada uma desvantagem da memória flash. Estudos como [2, 5, 19, 43] propõem formas de minimizar este problema.

A assimetria de leitura/escrita mostrou-se uma das principais vantagens da memória flash e também foi tratada com frequência nos estudos selecionados [11, 14, 16,40]. Operações de leitura na memória flash chegam a ser centenas de vezes mais ágeis do que em HDD, além de que, na memória flash a velocidade de acesso aleatório é próxima ao sequencial [14]. Esses são fatores que impactam no incremento de desempenho, principalmente em aplicações de leitura intensiva.

Outra característica importante do armazenamento flash é o paralelismo interno. Para possibilitar a utilização do paralelismo foram necessárias alterações de alguns algoritmos internos do SGBD, como demonstrado nos trabalhos de [1, 20, 22, 38]. Além disso, em [2] é proposto alterações em funções de estimativas de custos, possibilitando assim que durante o processamento de consulta fosse possível à escolha de métodos de acesso que permitam o uso do paralelismo.

Quanto à segunda questão, diferentes iniciativas e técnicas no contexto de processamento de consultas, page layout e indexação estão sendo criadas e/ou adaptadas, com vistas à especificidade da memória flash.

No contexto de processamento de consultas, essas técnicas foram divididas em três categorias. Com um maior número de ocorrências, os algoritmos de junção, priorizam estratégias de recuperar somente as colunas necessárias a junção como [15, 18, 20], e/ou estratégias que utilizam layout de armazenamento PAX [16, 17,36], como forma de minimizar o acesso a dados durante a junção de relações**.** Com um número menor de estudos, a iniciativa de criação de novos modelos de custo, têm por objetivo melhorar as estimativas de custos para operações de leitura/escrita sequencial e aleatória, bem como novos algoritmos para explorar operações de varredura paralela e junções, como trabalhos de [21,22].

Outra área frequentemente tratada nos estudos é a *page layout*. Iniciativas nessa área propõem novos *page layout* de armazenamento, visando principalmente tratar o problema de pequenos *random writes* decorrentes da atualização de dados e metadados. Neste sentido, verifica-se nos trabalhos de [15, 16 e 17] que a utilização de novos *page layout* possibilitou incremento no desempenho, por reduzir a quantidade de operações de escrita e leitura durante a execução de junções de relações.

Em contexto de indexação, três categorias foram identificadas. As duas mais presentes foram às técnicas baseadas na presença ou ausência de uma FTL. A terceira técnica utiliza armazenamento *hashing*. Desta forma, todas essas três técnicas priorizam minimizar o problema de sobrecarga de escrita, principalmente na atualização de índices.

Por fim, após analisar todos os 25 estudos selecionados, verifica-se que SGBDs atuais ainda têm um longo caminho a seguir na adequação de seu projeto às características únicas da memória flash. Além disso, destaca-se que alterações internas nos SGBDs não podem deixar de considerar as características dos HDDs, pois, no momento, o uso das duas tecnologias de armazenamento acontece em concomitância.

Também constata-se que as novas técnicas para processamento de consulta, *page layout* e indexação, mesmo que desenvolvidas para solução de problemas específicos ou para explorar novos recursos da memória flash, estão cada vez mais integradas, como pode ser observado na tabela 4, onde verifica-se, por exemplo, que para a proposta de novos algoritmos de junções, foram utilizados também novas técnicas de *page layout*. Outro exemplo é a proposta de novos modelos de custos, que por sua vez, utilizam técnicas de paralelismo para tal [22]. Também constatou-se que foram poucos os estudos a proporem novos modelos de custos *flash-aware*, O baixo de número de estudos (apenas 2) nesta área deve-se principalmente à necessidade de novos algoritmos flash-aware para operações como: varredura, junção e indexação.

Assim, este artigo apresentou um estudo sobre processamento de consulta, *page layout* e indexação em SGBDs que utilizam memória flash. Uma vez que a maioria dos projetos de SGBDs não têm acompanhado a evolução das tecnologias de armazenamento e deixam de explorar seu potencial, o tema desta pesquisa se mostra de primeira importância, até porque a memória flash cada vez mais se põe como alternativa ao HDD. Adicionalmente, esta revisão sistemática reuniu estudos relevantes sobre as questões de pesquisa definidas na seção 1, oferecendo, assim, uma base de referência para o desenvolvimento de novas pesquisas.

6. Referências

[1] Lee, E., et al. Optimizing Index Scans on *Flash* Memory SSDs. SIGMOD Record, Dez. 2011.

[2] Wu, C.-H., Kuo, T.-W., and Chang, L.-P. 2007. An efficient B-tree layer implementation for *flash*memory storage systems. ACM Trans. Embedd. Comput. Syst. 6, 3, Article 19 (July 2007), 23 pages. DOI = 10.1145/1275986.1275991 http://doi.acm.org/ 10.1145/1275986.1275991.

[3] Roh, Hongchan; Park, S., Kim, S., Shin, M., Lee,S., B+-tree Index Optimization by Exploiting Internal Parallelism of *Flash*-based Solid State Drives. 38th International Conference on Very Large Data Bases, August 27th 31st, 2012, Istanbul, Turkey.

[4] Agrawal, D., Ganesan, D., Sitaraman, R., Diao, Y., Singh S., Lazy-Adaptive Tree: An Optimized Index Structure for *Flash* Devices. VLDB '09, August 24-28, 2009, Lyon, France

[5] Li, Y., et al. Tree Indexing on Solid State Drives. 36th International Conference on Very Large Data Bases, 13-17 setembro, 2010.

[6] S. Nath and A. Kansal, "*Flash*DB: Dynamic self-tuning database for NAND *flash*," in Proc. ACM Int. Conf. Inform. Process. Sensor Netw.(IPSN), 2007, pp. 410–419.

[7] Jiang, Z., et al. AB-Tree: A Write-optimized Adaptive Index Structure on Solid State Disk. 11th Web Information System and Application Conference. 2014.

[8] Fang, H., et al. An Adaptive Endurance-Aware B+-Tree for *Flash* Memory Storage Systems. 2013.

[9] On, S. T., et al. *Flash*-Optimized B+-Tree. Journal of computer science and technology, maio, 2010.

[10] Ahn, J. , Kang, D., Jung, D., Kim, J., u*-Tree: An Ordered Index Structure for NAND *Flash* Memory with Adaptive Page Layout Scheme. IEEE TRANSACTIONS ON COMPUTERS, VOL. 62, NO. 4, APRIL 2013.

[11] Wang, L. e Wang, H. A New Self-adaptive Extendible Hash Index for *Flash*-based DBMS. 2010.

[12] Byun, S. e Hur, M. An index management using CHC-cluster for *flash* memory databases. The Journal of Systems and Software, 2008.

[13] Wu, K., et al. Multi-Level Bitmap Indexes for *Flash* Memory Storage. IDEAS, agosto 16-18. 2010.

[14] Rizvi, S., Chung., T., *Flash* Memory SSD based DBMS for Data Warehouses and Data Marts. Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conference.

[15] Evangelista, N. L. e Filho, J. A. Bt-Join: A Join Operator for Asymmetric Storage Device. SAC. Salamanca, Spain, 13-17 abril, 2015.

[16] Shah, M. A., et al. Fast Scans and Joins using *Flash* Drives. Proceedings of the Fourth InternationalWorkshop on Data Management on New Hardware, 13 junho, 2008.

[17] Tsirogiannis, D., et al. Query Processing Techniques for Solid State Drives. SIGMOD 09, 29 jun - 2 jul, 2009.

[18] Li, Y., et al. Optimizing Nonindexed Join Processing in *Flash* Storage-Based Systems. 2012.

[19] Na, G., Lee, S., Moon, B., Dynamic In-Page Logging for Bþ-tree Index. IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 24, NO. 7, JULY 2012.

[20] Lai, W., et al. Scan and Join Optimization by Exploiting Internal Parallelism of *Flash*-Based Solid State Drives. Springer-Verlag Berlin Heidelberg, pag 381-392, 2013.

[21] Bausch, D., et al. Making Cost-Based Query Optimization Asymmetry-Aware. New Hardware, 21 maio, 2012.

[22] Ghodsnia, P., Bowman, I., Nica, A., Parallel I/O Aware Query Optimization. SIGMOD'14, June 22–27, 2014, Snowbird, UT, USA.

[23] Ash, S., Lin, K. Optimizing database index performance for solid state drives. Proceedings of the 18th International Database Engineering & Applications Symposium (IDEAS'14), 2014.

[24] G.-J. Na, B. Moon, and S.-W. Lee, "In-Page Logging B-Tree for *Flash* Memory" Proc. 14th Int'l Conf. Database Systems for Advanced Applications (DASFAA '09), pp. 755-758, Apr. 2009.

[25] J.P. Higgins, S. Green, Cochrane Handbook for Systematic Reviews of Interventions, Version 5.1.0 [updated Março 2011], The Cochrane Collaboration, 2011, <www.cochrane-handbook.org>.

[26] Silberschatz, A., Henry, S.Sudarshan, S. Database Systems. 2012.

[27] B. A. Kitchenham. Guidelines for performing Systematic Literature Reviews in Software Engineering. EBSE Technical Report, EBSE-2007-01.

[28] Agrawal, N., Prabhakaran, V., Wobber, T., Davis, J.D., Manasse, M., Panigrahy, R.: *Design Tradeoffs for SSD Performance.* In: USENIX, pp. 57–70 (2008)

[29] Dirik, C., Jacob, B.: The performance of PC solid-state disks (SSDs) as a function of bandwidth, concurrency, device, architecture, and system organization. In: ISCA, pp. 279–289 (2009).

[30] Rosenblum, Mendel and Ousterhout, John K, "The Design and Implementation of a Log-Structured File System", Berkeley, CA: ACM, 1991.

[31] Lee, S., Na, G., Kim, J., Oh, J., Kim, Research Issues in Next Generation DBMS for Mobile Platforms. HCI'07, September 9-12, 2007.

[32] S.L. Pfleeger, Soup or art? The role of evidential force in empirical software engineering, IEEE Software 22 (1) (2005) 66–73.

[33] A. Ailamaki, D.J. DeWitt, M.D. Hill, and M. Skounakis, "Weaving Relations for Cache Performance," Proc. 27th Int'l Conf. Very Large Data Bases (VLDB), pp. 169-180, 2001.

[34] G.-J. Na, B. Moon, and S.-W. Lee, "In-Page Logging B-Tree for *Flash* Memory," Proc. 14th Int'l Conf. Database Systems for Advanced Applications (DASFAA '09), pp. 755-758, Apr. 2009.

[35] D. Kang, D. Jung, J.-U. Kang, and J.-S. Kim, u -Tree: An ordered index structure for NAND *flash* memory", in Proc. ACM Int. Conf. Embedded Softw. (EMSOFT), 2007, pp. 144–153.

[36] Ma, D., Hanf W. An Equi-join Algorithm Based on Low-update Conditions. System Science, Engineering Design and Manufacturing Informatization (ICSEM), 2011 International Conference on, *Guiyang, 2011, pp. 245-248*.

[37] Jin, et al. Read/write-optimized tree indexing for solid-state drives. The VLDB Journal Issue 5/2016.

[38] Shin et al. Optimizing Hash Partitioning for Solid State Drives. SAC 2016.

[39] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, K. El Emam, J. Rosenberg, Preliminary guidelines for empirical research in software engineering, IEEE Transactions on Software Engineering 28 (8) (2002) 721–734.

[40] I. Petrov, R. Gottstein, and S. Hardock. DBMS on modern storage hardware. In ICDE, pages 1545–1548, 2015.

[41] M. A. H. Chowdhur., & K.-H. Kimy. (2008). A Survey of *Flash* Memory Design and Implementation of Database in *Flash* Memory, in 3rd International Conference on Intelligent System and Knowledge Engineering, South Korea, pp. 1256 - 1259.

[42] Wang, Jiang-Tao ; Lai, Wen-Yu ; Meng, Xiao-Feng. *Flash*-Based Database: Studies, Techniques and Forecasts. In: Chinese Journal of Computers, 08/20/2013, Vol.36(8), pp.1549-1567.

[43] Roh, H. Kim W. Kim S. Park, S. A B-Tree index extension to enhance response time and the life cycle of *flash* memory. Information Sciences,Volume 179, Issue 18, 2009.

[44] Luo, T. et al., 2012. hStorage-DB: Heterogeneity-aware Data Management to Exploit the Full Capability of Hybrid Storage Systems, In: Proceedings of the Very Large Databases, Istanbul, Turkey.

[45] J. Gray and B. Fitzgerald. *Flash* disk opportunity for server applications. Queue, 6(4):18-23,2008.

[46] Kim, J., Seo, S., Jung, D., Kim, J.-S., and Huh., J. (2012). Advances in flash memory ssd technology for enterprise database applications. IEEE Transactions on Computers, 61:636–649.

[47] Ramakrishnan, R.; Gehrke, J. Sistemas de Gerenciamento de Banco de Dados. 3.ed. São Paulo: McGraw Hill, 2008.

[48] B. A. Kitchenham. Procedures for Performing Systematic Reviews, Technical Report TR/SE-0401 (Keele University, 2004).

[49] B. A. Kitchenham. O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey and S. Linkman, Systematic literature reviews in software engineering - A systematic literature review, Information and Software Technology, 51(1), (2009) 7–15.

ANEXO A - *BENCHMARK* TPC-H

A.1 *BENCHMARK* TPC-H

O *benchmark* TPC-H, concebido e distribuído pelo *Transaction Processing Performance Council* (TPC), compreende um conjunto de consultas ad hoc direcionadas para os negócios e alterações de dados simultâneos. Este *benchmark* avalia e simula o desempenho de um de Data Warehouse. Denomina-se Data Warehouse um imenso armazém de informações coletadas de variadas fontes e que gera dados para gerenciamento auxiliando na tomada de decisões.

Uma estrutura padrão formada por oito tabelas é utilizada para realizar os testes do método TPC-H. São usadas duas tabelas de fatos (Lineitem e Orders) e seis dimensionais (Nation, Region, Supplier, Part, Customer e Partsupp).

Os valores das tabelas de fatos medem o desempenho do negócio e se encontram na parte central do modelo dimensional (este modelo é utilizado para apoio à decisão). Estas tabelas armazenam nas tabelas dimensionais as chaves para as características correlatas e os fatos ocorridos. As tabelas de dimensão organizam de forma geral a informação corporativa, elas são as áreas do negócio e provêm inúmeras perspectivas dos dados. Para assegurar a exatidão dos dados, as consultas ocorrem primeiramente nas tabelas de dimensão e após nas tabelas de fatos.

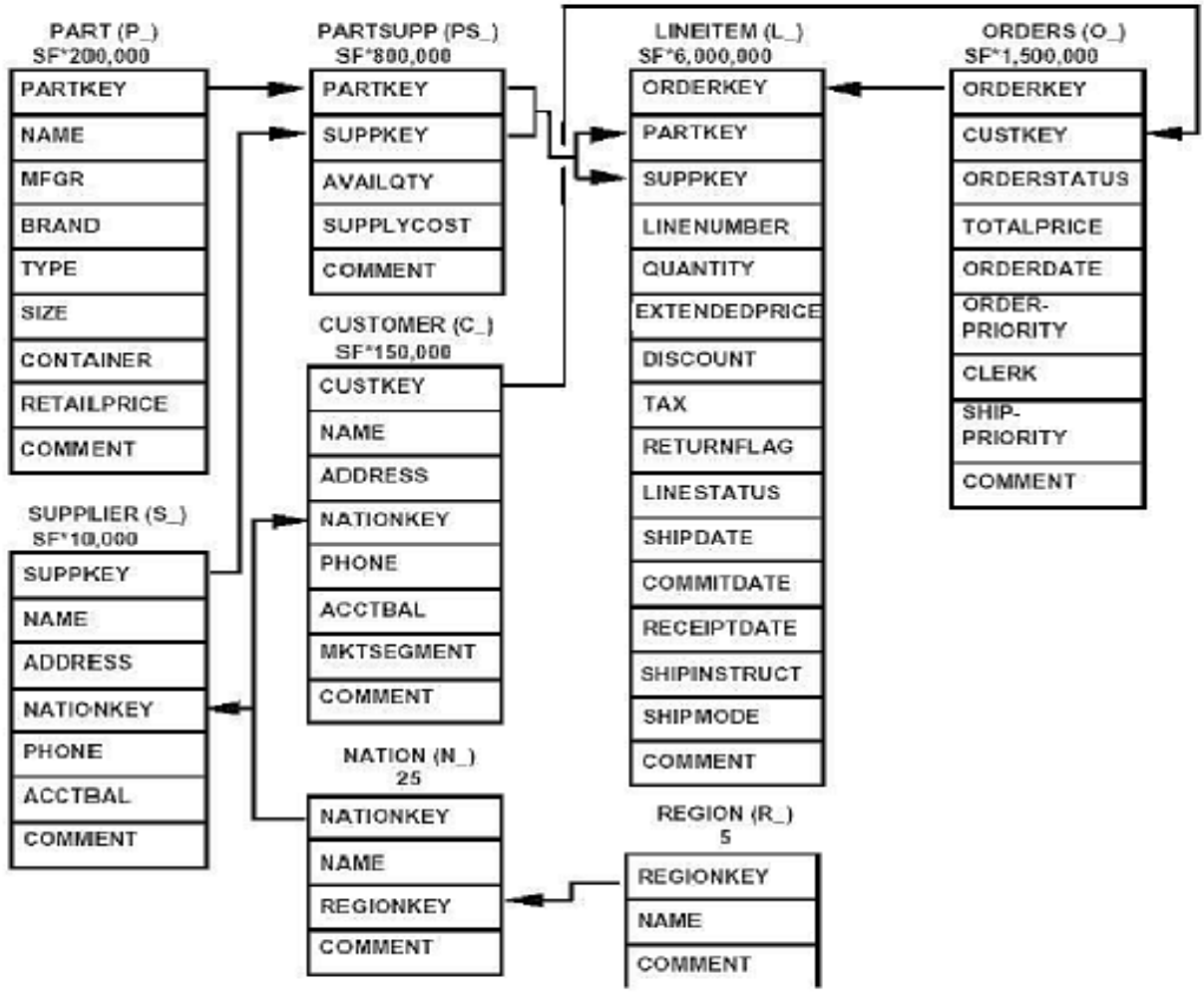
A figura A.1 apresenta o modelo relacional do *benchmark* TPC-H. Segue importantes observações referentes à análise da figura A.1 (MORELLI, 2006):

- A cardinalidade é representada pelo número que aparece abaixo do nome da tabela. Podendo ser variável ou fixa (tabelas nation e region), no caso de variável multiplica-se por um scale fator, definido na criação da base, por uma constante. Exemplificando, caso um scale fator possui o valor 5, então existirá na tabela de clientes (customer) o total de 750.000 tuplas;
- São exibidas cinco regiões (continentes), que reúnem vinte e cinco nações (tabelas region e nation). Os fornecedores (tabelas supplier e customer) e os clientes estão vinculados às nações. Os clientes efetuam os pedidos de compras (tabela orders) e os fornecedores provê os componentes (tabela part) de itens de compra. Um fornecedor pode disponibilizar inúmeros itens e um item pode ser oferecido por inúmeros fornecedores, esta relação N x N é armazenada na tabela partsupp. Concluindo, a tabela mais numerosa do modelo (lineitem) interliga itens de compra a pedidos;
- O prefixo utilizado para denominar os campos de uma tabela está localizado nos parênteses próximos ao nome da tabela. Assim, é chamada S SUPPKEY a chave primária da tabela de fornecedores;

- As flechas mostram as associações entre chaves estrangeiras e primárias. Desta forma, podemos verificar que a chave primária da tabela nation encontra-se associada ao campo nationkey na tabela de fornecedores (supplier).

O fator de escalabilidade (SF - scale factor) define o tamanho total da base de dados. Considerando SF=1, a base total utiliza cerca de 1 GB e os volumes de cada tabela são apresentados na Figura A.1. Considerando SF=50, neste caso, a base irá ocupar 50 GB e a tabela de itens de pedidos de compra conterá 300 milhões de tuplas (6 milhões x 50).

Figura A.1. Esquema Relacional do TPC-H



Fonte: Monteiro (2008).

O acesso à base de dados se dá por um conjunto de consultas que possui características ad hoc, assim não se conhece os parâmetros e nem a ordem de execução das duas funções (uma função insere e a outra elimina os dados) e de cada uma das 22 consultas (leitura).

Conforme é apresentado no quadro A.1, o *benchmark* TPC-H é composto por inúmeros testes. Inicialmente, popula-se o banco de dados com os arquivos do tipo texto gerados anteriormente pela ferramenta DBGEN (Database Generator) de acordo com as diretrizes do TCP-H. As vinte e duas consultas e seus parâmetros são lidos em uma ordem definida pelo programa gerador QGEN *(Query Generator).*

O número de fluxos de comandos usados no teste Throughput deve se adequar ao fator de escalabilidade da base de dados. A especificação oficial do *benchmark* TPC-H mostra a seguinte correlação (quadro A.2): o teste Throughput gera duas medidas, o tempo transcorrido e a vazão (quantidade de comandos por segundo).

As funções de atualização (RF1, RF2) operam sobre as tabelas maiores (orders e lineitem), onde a primeira insere e a segunda função exclui as tuplas. Convém observar que o número de tuplas permanece estável devido à execução de uma RF sempre vir seguida da outra, visto que a RF1 insere 0,1% de tuplas e a RF2 exclui outras 0,1% de tuplas.

Quadro A.1. Testes do *Benchmark* TPC-H

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

Quadro A.2. Números de Fluxos usados no teste *Throughput*

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

A RF1 e a RF2 obtêm por argumento o fator de escalabilidade de acordo com o tamanho atual das tabelas. Desta forma, na tabela lineitem, se SF=1, a execução de RF1 ocasionará a inclusão de seis mil tuplas (6.000.000 x 0,01). Caso haja necessidade de aumentar o percentual de tuplas atingidas, as funções deveriam ser executadas considerando SF=30, para bases com fator de escalabilidade um, haveria o aumento de trinta vezes mais, de 0,1% para 3%, gerando assim 180.000 tuplas.

A.1.1. OLAP

OLAP (On-Line Analytical Processing) consiste em uma tecnologia que possibilita aos executivos, analistas de negócios e gerentes analisar e visualizar os dados corporativos de forma ágil, interativa e consistente.

Esta tecnologia realiza uma análise multidimensional e dinâmica dos dados consolidados de uma organização possibilitando ao usuário final realizar atividades analíticas e navegacionais. OLAP é uma tecnologia implementada em ambiente cliente/servidor e multiusuário, ela disponibiliza respostas rápidas as consultas *ad hoc*.

OLAP ajuda o usuário a sumarizar dados corporativos através de análises históricas, visões comparativas e personalizadas, projeções e também elaborações de cenários.

A.1.2. As cargas de trabalho do *benchmark* TPC-H

Segue as consultas *ad hoc* que compõem o *benchmark* TPC-H:

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

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

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

    4.  SELECT o_orderpriority, count(*) as order_count
        FROM orders
        WHERE o_orderdate >= date '19980801'
```

```
        and o_orderdate < date '19980808' + interval '3 month'
        and exists ( SELECT *
                FROM lineitem
                WHERE l_orderkey = o_orderkey
                                and l_commitdate < l_receiptdate
                )
                GROUP BY o_orderpriority
                 ORDER BY o_orderpriority
```

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

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

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

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

9. SELECT nation, o_year, sum(amount) as sum_profit
   FROM (SELECT n_name as nation, extract(year from o_orderdate) as o_year, l_extendedprice * (1 - l_discount) –
                ps_supplycost * l_quantity as amount

```
            FROM part, supplier, lineitem, partsupp, orders, nation
            WHERE s_suppkey = l_suppkey and ps_suppkey = l_suppkey
                    and ps_partkey = l_partkey and p_partkey = l_partkey
                    and o_orderkey = l_orderkey and s_nationkey = n_nationkey
                    and p_name like '%blush%'
          ) as profit
    GROUP BY nation, o_year
    ORDER BY nation, o_year desc

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

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

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

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

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

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

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

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

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

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


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

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


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


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