

CURSO DE CIÊNCIA DA COMPUTAÇÃO

Eduardo Weiland

**AMBIENTE DE RECOMENDAÇÃO DE ÍNDICES PARA BANCOS DE DADOS
MYSQL**

Santa Cruz do Sul

2016

Eduardo Weiland

**AMBIENTE DE RECOMENDAÇÃO DE ÍNDICES PARA BANCOS DE DADOS
MYSQL**

Trabalho de Conclusão apresentado ao Curso de Ciência da Computação da Universidade de Santa Cruz do Sul, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Msc. Eduardo Kroth

Santa Cruz do Sul
2016

RESUMO

Ferramentas para recomendação de índices são utilizadas para auxiliar na definição dos índices que devem ser criados em um banco de dados relacional, visando obter um melhor desempenho para execução de consultas. Vários bancos de dados já oferecem ferramentas com esse objetivo, como Microsoft SQL Server, Oracle Database, IBM DB2 e PostgreSQL. O presente trabalho apresenta o desenvolvimento de um ambiente para recomendação de índices para bancos de dados MySQL. O ambiente analisa uma carga de trabalho composta de diversas consultas SQL. Essa carga de trabalho é carregada para a ferramenta a partir de um arquivo XML em um formato pré-definido. Cada consulta é interpretada e um conjunto de índices candidatos será gerado. Os índices são, então, criados em um banco de dados configurado pelo usuário do ambiente, que já deve conter todas as tabelas e dados necessários. Os índices candidatos são avaliados através de instruções EXPLAIN, calculando o custo de todas as consultas utilizando cada índice possível. A saída ao final da execução da ferramenta é o conjunto de índices recomendados que oferece o menor custo total para a carga de trabalho. Os índices selecionados ao final da execução da ferramenta apresentaram um ganho de performance considerável, em relação à inexistência de índices no banco de dados. Todavia, o resultado obtido não foi conclusivo quando comparado aos índices já existentes na base de dados utilizada, pois todas as soluções geradas já eram utilizadas pelo sistema analisado.

Palavras-chave: MySQL, recomendação de índices, modelo físico.

ABSTRACT

Index advisor tools are used to assist in the definition of indexes that should be created in a relational database in order to obtain better performance for running queries. Several databases already offer tools for this purpose, such as Microsoft SQL Server, Oracle Database, IBM DB2 and PostgreSQL. This paper proposes the development of an index advisor environment for MySQL databases. The environment analyzes a workload consisting of several SQL queries. This workload is loaded into the tool from an XML file in a predefined format. Each query is interpreted and a set of candidate indexes is generated. The indexes are then created in a database configured by the user, which should already contain all the tables and data required. Candidate indexes are evaluated by EXPLAIN statements, calculating the cost of all queries using every possible index. The output at the of the execution is a set of recommended indexes that offers the lowest total cost to the workload. The resulting set of indexes showed a considerable performance gain compared to the absence of indexes in the database. However, the result was not conclusive when compared to the existing indexes in the database that was used to run the tests, as all the generated solution were already used by the system.

Keywords: MySQL, index advisor, physical model.

LISTA DE ILUSTRAÇÕES

Figura 1	Modelo de comunicação cliente/servidor em bancos de dados relacionais.	12
Figura 2	Etapas do processamento de consultas em um SGBD relacional.....	15
Figura 3	Tempo de acesso ao disco utilizando índice <i>B+tree</i> vs. acesso sequencial.	19
Figura 4	Arquitetura do sistema desenvolvido no trabalho de Alagiannis et al.....	21
Figura 5	Interface da ferramenta <i>Microsoft SQL Server DTA</i>	22
Figura 6	Interface da ferramenta <i>DB2 Design Advisor</i>	23
Figura 7	Resultados obtidos por Zilio et al. utilizando um banco de dados TPC-H.....	24
Figura 8	Utilização do ambiente desenvolvido.....	27
Figura 9	Arquitetura interna do ambiente desenvolvido.....	28
Figura 10	Processo de geração de índices candidatos.	34
Figura 11	Interface do MIST exibindo índices candidatos gerados.....	36
Figura 12	Processo de verificação dos índices candidatos.....	37
Figura 13	Fluxograma para escolha dos índices da solução final.....	40
Figura 14	Modelo do banco de dados utilizado para validação da ferramenta.....	42
Figura 15	Índices candidatos gerados com o banco de dados de validação.	44
Figura 16	Índices recomendados pelo MIST para o banco de dados de validação.	45

LISTA DE TABELAS

Tabela 1	Comparativo de trabalhos relacionados.....	25
Tabela 2	Número de registros por tabela no banco de dados de validação.....	41
Tabela 3	Contribuição do trabalho desenvolvido.....	46

LISTA DE CÓDIGOS

Código 1 Exemplo de arquivo no formato MSDF.	29
Código 2 Exemplo de arquivo no formato MQLF.	31
Código 3 Exemplo de saída do EXPLAIN no formato JSON.	37
Código 4 Exemplo de consulta SQL escolhida para os testes.	43

LISTA DE ABREVIATURAS

API	<i>Application Programming Interface</i>
DBA	<i>Database System Administrator</i>
DDL	<i>Data Definition Language</i>
DML	<i>Data Manipulation Language</i>
JDBC	<i>Java Database Connectivity</i>
JSON	<i>JavaScript Object Notation</i>
MDC	<i>Multidimensional Clustering</i>
ODBC	<i>Open Database Connectivity</i>
SGBD	Sistema de Gerenciamento de Banco de Dados
SQL	<i>Structured Query Language</i>
XML	<i>eXtensible Markup Language</i>

SUMÁRIO

1	INTRODUÇÃO	9
2	ARQUITETURA DE BANCOS DE DADOS	12
2.1	Modelo relacional.....	13
2.2	Aplicação cliente.....	13
2.3	Interface de comunicação.....	14
2.4	Processamento de consultas	14
3	ESTRATÉGIAS DE OTIMIZAÇÃO	17
3.1	Índices	17
3.2	Outras otimizações.....	19
4	TRABALHOS RELACIONADOS	21
4.1	Comparativo entre trabalhos relacionados	24
5	SOLUÇÃO DESENVOLVIDA – MIST (<i>MYSQL INDEX SUGGESTION TOOL</i>) .	26
5.1	Modificação da solução proposta.....	26
5.2	Arquitetura interna.....	27
5.3	Entrada de dados	28
5.3.1	O formato do Arquivo de Definição do Modelo do Banco de Dados (MSDF).....	29
5.3.2	O formato do Arquivo de Histórico de Consultas (MQLF)	30
5.4	Geração de índices candidatos.....	33
5.5	Verificação dos índices candidatos.....	36
5.6	Geração de solução final.....	39
5.7	Validação da ferramenta desenvolvida	41
6	CONCLUSÃO	46
	REFERÊNCIAS.....	48

1 INTRODUÇÃO

É essencial para quase todos os *softwares* ter um bom desempenho, seja por necessidade dos clientes ou para oferecer uma melhor experiência para os usuários, esse é um fator cada vez mais importante em muitas aplicações, e diretamente relacionado à forma em que os dados são armazenados e acessados. Atualmente, os Sistemas de Gerenciamento de Banco de Dados (SGBDs) relacionais são um dos meios de armazenamento mais utilizados no desenvolvimento de novas aplicações (HEUSER, 2009).

A melhoria do desempenho de bancos de dados relacionais consiste em identificar quais são os principais problemas de performance e tentar resolvê-los (THALHEIM; TROPMANN, 2011). Os problemas mais comuns podem ser configurações de *hardware* do servidor inadequadas, consultas escritas de forma não otimizada, ou a organização física dos dados em disco não corresponder às necessidades da aplicação.

A modelagem física de um banco de dados é o processo que define a organização física dos dados em disco, e compreende a definição de índices, particionamento, agrupamento (*clustering*) e materialização de dados (LIGHTSTONE; TEOREY; NADEAU, 2007, p. 7). A otimização manual dessa modelagem é uma tarefa complexa e que exige muito conhecimento técnico por parte do administrador do banco de dados – *Database System Administrator* (DBA). Com isso, a necessidade de ferramentas que automatizem esse processo é cada vez mais importante (ALAGIANNIS et al., 2010).

A otimização do modelo físico é vista pelos autores da área como a estratégia mais eficiente para melhorar o desempenho dos SGBDs (THALHEIM; TROPMANN, 2011; ZILIO et al., 2004). Segundo Petraki, Idreos e Manegold (2015), dentre as possíveis melhorias no esquema físico de um banco de dados, a escolha de índices adequados ainda é a que apresenta os melhores resultados. Algumas das técnicas empregadas para otimização de índices mais conhecidas e aplicadas são *offline*, *online* e adaptativa.

A indexação *offline* foi a primeira estratégia adotada pelos SGBDs comerciais. Consiste em uma ferramenta que auxilia o trabalho do DBA, analisando uma determinada carga de trabalho (*workload*) e sugerindo melhorias. Porém, essa técnica requer o conhecimento prévio da carga de trabalho a qual o servidor deve estar preparado, o que nem sempre é possível, visto a dinamicidade das aplicações modernas.

Por sua vez, a indexação *online* surgiu como uma alternativa para resolver o maior problema da indexação *offline*, sugerindo uma abordagem em que a carga de trabalho do servidor

é monitorada continuamente. Conforme a carga de trabalho sofre alterações, novos índices podem ser criados e índices já existentes podem ser atualizados ou excluídos conforme a necessidade. No entanto, tais operações requerem um longo tempo para completar e consomem mais recursos de processamento, tornando-se inviáveis para bancos com um maior volume de dados.

A estratégia de indexação adaptativa aparece como uma opção mais econômica no uso de recursos, enquanto ainda resolve o problema de não ter um conhecimento prévio da carga de trabalho. Essa técnica consiste em realizar pequenas atualizações de forma incremental nos durante a execução de cada consulta. Dessa forma, os índices se mantêm sempre atualizados e instantaneamente preparados para a carga de trabalho à qual o servidor é submetido.

Os trabalhos desenvolvidos mais recentemente sugerem, geralmente, uma abordagem *online* ou adaptativa, ou ainda uma nova aproximação, como a abordagem holística proposta por Petraki, Idreos e Manegold (2015). Poucos trabalhos continuam explorando a técnica de indexação *offline*. No entanto, em aplicações onde a carga de trabalho do banco de dados é razoavelmente constante e previsível, as técnicas de otimização *online* e adaptativa são desnecessárias.

Nessas condições, os sistemas obteriam maiores benefícios tendo um modelo físico definitivo adaptado e otimizado à maioria das condições de operação normal da aplicação. Isso auxiliaria na redução de custos de manutenção do banco de dados ao longo do tempo, além de evitar estressar o sistema continuamente com as verificações de performance do modelo físico realizadas pelas técnicas de otimização mencionadas acima. Dessa forma, uma abordagem *offline* ainda pode ser aplicada em casos específicos.

O objetivo geral desse trabalho é desenvolver um ambiente de código aberto para recomendação de índices para bancos de dados MySQL, visando obter um conjunto de índices equilibrado que oferece um bom desempenho geral para um determinado conjunto de consultas.

Os objetivos específicos do presente trabalho são:

- Verificar quais as principais características que influenciam a performance dos bancos de dados relacionais em geral;
- Definir um modelo de avaliação de performance do SGBD para comparar os resultados obtidos com a solução desenvolvida e as ferramentas já existentes no mercado;
- Elaborar a ferramenta de forma que possibilite o desenvolvimento futuro de integração com diferentes SGBDs;
- Validar a solução desenvolvida em um ambiente de uso real.

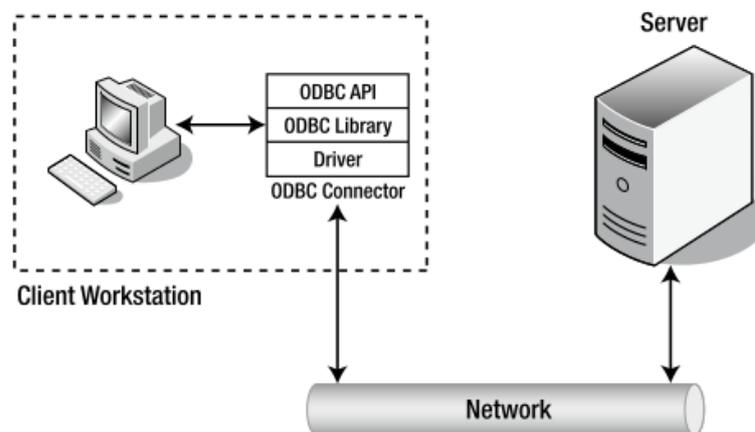
O trabalho é estruturado da seguinte forma: no capítulo 2 são descritos os principais componentes que formam a arquitetura geral de um banco de dados relacional; o capítulo 3 introduz alguns aspectos importantes relacionados à otimização de um banco de dados; o capítulo 4 apresenta alguns trabalhos relacionados e realiza uma comparação entre esses e o presente trabalho; já no capítulo 5 é apresentada a solução que foi desenvolvida. O trabalho encerra apresentando os resultados e conclusões obtidos e as referências utilizadas.

2 ARQUITETURA DE BANCOS DE DADOS

A arquitetura básica de um banco de dados relacional é composta de um cliente e de um servidor, como pode ser visualizado na figura 1. O cliente possui poucas funcionalidades próprias, sendo utilizado apenas como interface entre o servidor e o usuário. O servidor, por outro lado, é responsável por todo o processamento das consultas. Essa separação entre cliente e servidor favorece a independência de dados, onde o cliente não precisa saber como os dados são fisicamente armazenados, simplificando o desenvolvimento de novas aplicações (BELL, 2012, p. 30).

A comunicação entre aplicações cliente e servidor geralmente é feita através de uma conexão de rede, mas também pode ser realizada por estruturas internas do sistema operacional (*sockets, pipes, etc.*) ou integrando-se o banco de dados à aplicação (SGBD embarcado). Qualquer que seja o meio de comunicação utilizado, o cliente envia consultas para o servidor, que a processa e retorna o resultado para o cliente.

Figura 1 – Modelo de comunicação cliente/servidor em bancos de dados relacionais.



Fonte: Bell (2012, p. 28)

Nas seções seguintes é descrita a estrutura interna de cada um dos elementos que compõem a arquitetura de um banco de dados relacional. A seção 2.1 descreve o modelo relacional proposto por E. F. Codd, no qual os SGBDs relacionais são baseados. Na seção 2.2 é abordado o funcionamento da aplicação cliente, bem como a forma de apresentação das consultas e dos resultados. A seção 2.3 descreve como é feita a comunicação entre as aplicações cliente e servidor, e as estruturas utilizadas para esta finalidade. Por fim, a seção 2.4 expõe as etapas de processamento das consultas no servidor, incluindo os estágios de análise léxica e sintática, a geração do plano de execução e a efetiva realização da operação requisitada pelo cliente.

2.1 Modelo relacional

O modelo de dados relacional proposto inicialmente por E. F. Codd em 1970 define o conceito de um método de armazenamento de dados que pode ser acessado e manipulado utilizando uma linguagem de consulta (BELL, 2012, p. 25). Devido à sua simplicidade e bom embasamento teórico e matemático, o modelo tornou-se o mais utilizado para armazenamento de dados estruturados (LIGHTSTONE; TEOREY; NADEAU, 2007).

No modelo relacional, os dados são representados como informações (atributos) sobre uma determinada entidade. Um registro ou tupla contém os valores para os atributos de uma entidade. As tabelas são um conjunto de registros que possuem os mesmos atributos, e podem ser relacionadas a outras tabelas utilizando-se restrições (*constraints*).

Não existe uma linguagem de consulta padrão definida pelo modelo relacional. Contudo, muitas aplicações optaram por implementar a *Structured Query Language* (SQL), uma linguagem de consulta que se assemelha com linguagem natural. Atualmente, a SQL é o padrão do mercado, sendo suportada pelos SGBDs mais conhecidos e utilizados (BELL, 2012, p. 26).

2.2 Aplicação cliente

Uma aplicação cliente pode ser qualquer *software* que se comunique com o SGBD para manipular os dados de alguma forma. Essa definição inclui tanto aplicações desenvolvidas para serem utilizadas por usuários finais quanto aplicações de gerenciamento e modelagem do banco de dados. Qualquer *software* cliente envia comandos para o servidor utilizando, geralmente, a linguagem SQL, recebe a resposta e exibe para o usuário.

Os comandos SQL podem ser divididos em duas categorias principais: *Data Definition Language* (DDL), utilizada para criação das estruturas de armazenamento dos dados, como tabelas e índices, e a *Data Manipulation Language* (DML), responsável pelas operações de manipulação dos dados armazenados, como criação de novos registros, modificação dos dados já existentes e recuperação dos dados salvos.

Cada comando SQL possui uma sintaxe própria e bem-definida. Todos possuem a especificação de uma ação, por exemplo criar uma tabela (CREATE TABLE) ou atualizar dados (UPDATE). Em seguida são informados demais parâmetros que afetam o tipo de estrutura que será criado ou os dados que serão atingidos.

2.3 Interface de comunicação

Existem, de maneira geral, duas diferentes formas de comunicação entre uma aplicação cliente e o SGBD. Em uma delas, o cliente e o servidor são processos separados, e a comunicação ocorre utilizando alguma interface de rede ou estruturas especiais oferecidas pelo sistema operacional, como *pipes*, *FIFOs* e *sockets*. Esse meio de comunicação é o mais comum e é utilizado na maioria das aplicações modernas. Utilizam o conceito de envio de consultas SQL para o servidor e o retorno da resposta para o cliente utilizando um protocolo de comunicação padronizado, geralmente o *Open Database Connectivity* (ODBC) ou uma de suas variantes, como o *Java Database Connectivity* (JDBC). Estes protocolos definem o formato das mensagens enviados entre cliente e servidor, além de promover uma forma de acesso padrão para diferentes SGBDs.

Outra forma de comunicação é o uso de um SGBD embarcado. Dessa forma, a aplicação cliente utiliza funções disponibilizadas em uma *Application Programming Interface* (API) fornecida pelo banco de dados para acessar diretamente os arquivos de armazenamento dos dados. O exemplo de banco de dados mais conhecido que oferece essa funcionalidade é o SQLite¹, porém outros bancos de dados também oferecem suporte a essa opção (BELL, 2012, p. 195). Mesmo nessa modalidade, as consultas ainda são escritas em SQL ou em outra linguagem suportada.

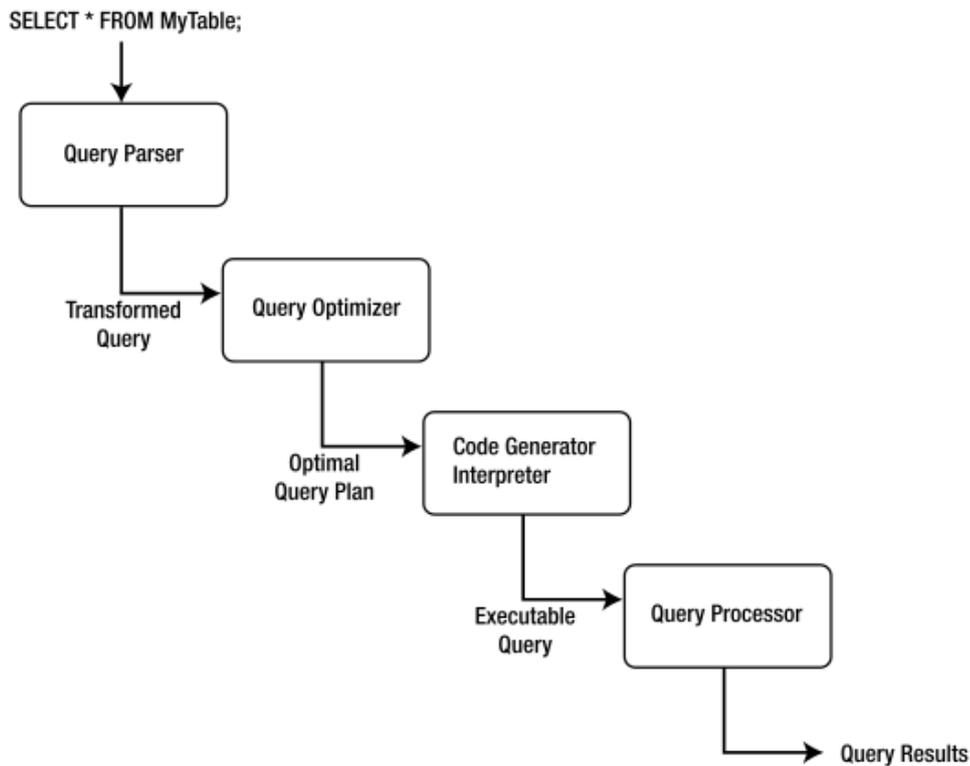
2.4 Processamento de consultas

Em bancos de dados que seguem o modelo cliente/servidor, o processamento das consultas é feito de forma sequencial no servidor e separado em várias etapas. Cada etapa é responsável por realizar alguma operação ou transformação sobre a consulta e gerar uma saída, que será utilizada na próxima etapa. Uma visão geral do processo de execução das consultas é exibido na figura 2.

A primeira tarefa executada quando uma nova consulta é recebida pelo servidor é a análise e interpretação do seu significado (*Query Parser*). Considerando uma consulta escrita em SQL, é necessário identificar os atributos e tabelas que o cliente deseja acessar, e quais as condições definidas para restringir os registros que devem ser retornados. Todas essas informações devem ser extraídas da consulta recebida e convertidas para uma estrutura com a qual o SGBD possa trabalhar nas etapas seguintes.

¹SQLite: <http://sqlite.org/>

Figura 2 – Etapas do processamento de consultas em um SGBD relacional.



Fonte: Bell (2012, p. 33)

A consulta é convertida para uma estrutura em árvore identificando os componentes da lógica relacional, isto é, quais atributos devem ser retornados (projeção), quais junções entre tabelas devem ser realizadas, quais critérios de seleção devem ser aplicados e, caso indicado na consulta, as funções de agregação e ordenação que devem ser aplicadas. Todas essas informações formam a estrutura gerada pela etapa de análise da consulta.

Após a criação da árvore de consulta, na etapa seguinte (*Query Optimizer*) são identificadas as possíveis abordagens para acesso aos dados e realização da junção entre tabelas. Essa tarefa é executada pelo otimizador, que é responsável por montar um plano de execução otimizado para a consulta a ser efetuada.

Existem quatro principais meios de otimização de consultas que são utilizados em SGBDs relacionais (BELL, 2012). 1) A otimização baseada em custo gera vários planos de execução equivalentes e então escolhe o que possui o menor custo, baseado em estatísticas coletadas em consultas anteriores. 2) A otimização heurística implementa algumas boas práticas para execução da consulta, utilizando algumas regras para eliminar os casos que provavelmente apresentariam um desempenho ruim e sugerindo opções que possivelmente apresentem alguma melhoria na performance.

A combinação dos métodos de otimização baseada em custo e heurísticas resulta na (3) otimização paramétrica, visando reduzir a quantidade de planos de execução a serem avaliados, através do uso de funções heurísticas para ignorar os piores planos. Uma quarta alternativa, que ainda não é implementada em SGBDs comerciais, é a (4) otimização semântica, que ainda está em pesquisa, cujo objetivo é que o otimizador tenha conhecimento do esquema do banco de dados e das restrições existentes, simplificando partes da consulta automaticamente.

A saída do otimizador é o plano de execução que foi escolhido de acordo com os parâmetros de performance como sendo o mais otimizado. Na etapa seguinte (*Code Generator Interpreter*), o plano é convertido em um algoritmo para acessar as estruturas físicas de armazenamento dos dados, como os índices e as tabelas. A etapa de execução da consulta (*Query Processor*) pode ser implementada utilizando duas estratégias: iterativa e interpretativa.

A estratégia interpretativa consiste em transformar o plano de execução em uma sequência de chamada de funções pré-compiladas. As funções utilizadas são implementadas de forma genérica e executam tarefas básicas do processamento da consulta, não sendo otimizadas para nenhum caso específico. É a mais utilizada em sistemas de bancos de dados relacionais (BELL, 2012, p. 35).

Na estratégia iterativa, o SGBD implementa funções para operações discretas (junção, projeção, etc.). O processamento da consulta consiste na transformação do plano de execução em um programa que utiliza essas funções disponibilizadas pelo banco de dados e subsequente compilação do mesmo em um arquivo binário executável.

Para recuperar as informações solicitadas, o SGBD precisa definir como será feito o acesso aos dados no disco. A estrutura mais utilizada nos bancos de dados modernos são arquivos, acessados por meio do sistema de arquivos do sistema operacional. Um motor de armazenamento (*storage engine*) estabelece a estrutura e forma de acesso aos arquivos, tendo como principal objetivo minimizar os custos de operações de entrada e saída (E/S). Com essa finalidade, alguns dos métodos mais empregados são o uso de estruturas que permitam acessar apenas as informações relevantes do disco, um mecanismo de *cache* ou *buffer* para aprimorar o tempo de leitura dos dados, e o uso de índices para auxiliar a obter os registros desejados.

3 ESTRATÉGIAS DE OTIMIZAÇÃO

Nas próximas seções são detalhadas algumas das estruturas e técnicas utilizadas pelos DBAs quando se deseja obter um melhor desempenho na execução das consultas em bancos de dados relacionais. Na seção 3.1 é detalhado o assunto de indexação, mencionando os vários tipos de índices existentes e como são utilizados. A seção 3.2 descreve brevemente outras abordagens existentes para otimização de bancos de dados, como particionamento, visões materializadas e agrupamento de dados.

3.1 Índices

Um índice é uma estrutura de dados auxiliar utilizada pelo SGBD durante a execução de uma consulta para encontrar os dados mais rapidamente (LIGHTSTONE; TEOREY; NADEAU, 2007, p. 53). O uso de índices como forma de localizar dados específicos em um banco de dados é uma das formas mais comuns de otimização e também a que apresenta o melhor resultado quando os índices corretos são selecionados (PETRAKI; IDREOS; MANEGOLD, 2015). De forma geral, os índices podem ser classificados em simples ou compostos, primários ou secundários e únicos ou não-únicos.

Os índices simples são formados por apenas um atributo da tabela. Dessa forma, a chave do índice é o valor do atributo indexado, e o índice informa a localização de um ou mais registros da tabela que possuem o mesmo valor da chave para esse atributo. Os índices compostos são criados para múltiplos atributos de uma mesma tabela. A chave do índice, nesse caso, é formada pela concatenação dos valores dos atributos indexados, e indica a localização de registros que possuam os mesmos valores concatenados da chave para os mesmos atributos. Os índices compostos apresentam um desempenho melhor do que o uso de índices simples quando os dados são filtrados por mais de um atributo presente no índice, porém sua utilização é restrita apenas às consultas que se encaixam em alguns critérios (LIGHTSTONE; TEOREY; NADEAU, 2007, p. 21). A ordem dos atributos no índice é importante, uma vez que a chave é formada pela concatenação dos valores, as consultas devem incluir os atributos que estão no começo da chave para utilizar o índice.

Os índices primários definem a organização principal dos dados na tabela. A chave desse tipo de índice, também conhecida como chave primária, deve ser um valor único para cada registro. Esse é um dos índices mais importantes em uma banco de dados, pois a maioria das junções entre tabelas é feita através da chave primária (LIGHTSTONE; TEOREY; NADEAU,

2007, p. 56–57). Cada tabela pode possuir um único índice primário, e todos os outros índices da tabela são chamados de secundários.

Índices únicos são formados por atributos cujos valores não podem ser repetidos em toda a tabela. Todos os índices primários devem ser índices únicos, mas índices secundários também podem ser definidos como tal. Cada entrada em um índice único identifica exclusivamente um registro na tabela, enquanto que em índices não-únicos para cada valor da chave podem existir vários identificadores de linhas na tabela.

Os SGBDs relacionais oferecem diferentes estruturas para definição dos índices, sendo as principais e mais utilizadas: *B+tree*, *hash* e *bitmap*. Estas estruturas definem como os dados do índice são organizados em disco e acessados posteriormente na etapa de execução da consulta.

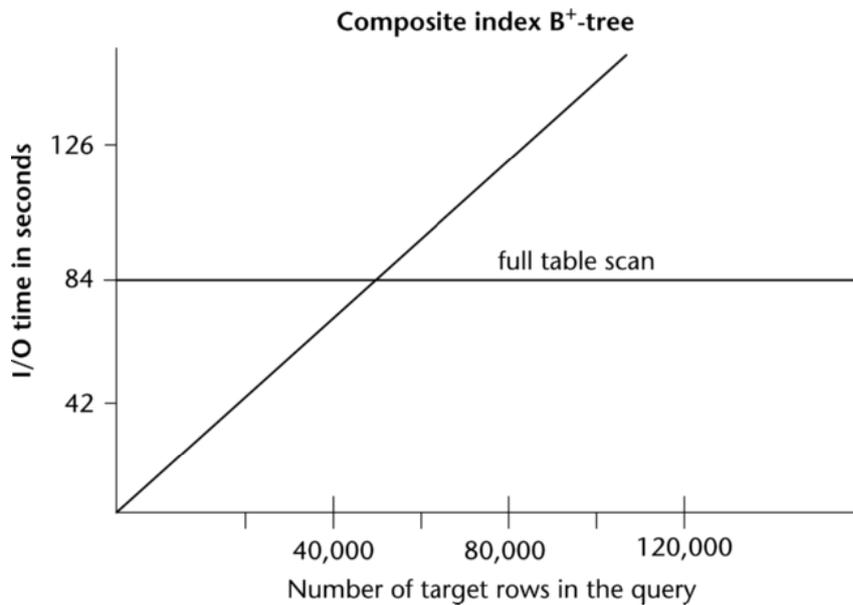
Os índices *B+tree* utilizam uma estrutura de árvore com ponteiros para blocos de dados. O tamanho da árvore varia de acordo com a quantidade de valores distintos existentes para os atributos que compõem o índice. Os nós intermediários da árvore possuem os valores dos atributos ordenados e ponteiros para sub-árvores de busca. Esses valores são utilizados para identificar qual sub-árvore contém a chave pesquisada, reduzindo o espaço de busca e possibilitando encontrar os nós folha de forma eficiente, com poucas operações de entrada/saída de dados.

Os nós folha dos índices *B+tree* possuem as informações realmente importantes: a chave, que são os valores dos atributos que compõem o índice, um identificador de um ou mais registros da tabela que possuem o valor da chave, a partir do qual o SGBD pode localizar precisamente a linha indicada, e um ponteiro para o próximo nó folha da árvore, que agiliza buscas por intervalos de valores, acessando diretamente os nós folha sem caminhar pela árvore.

O desempenho desse tipo de índice depende diretamente de sua seletividade, como mostra a figura 3. O tempo necessário para o acesso sequencial à tabela é constante, enquanto que o tempo para leitura dos dados utilizando o índice cresce à medida que mais registros são retornados pela consulta.

Os índices do tipo *bitmap* são utilizados para atributos com baixa seletividade, ou seja, com poucos valores diferentes. Cada possível combinação de valores para os atributos do índice possui um vetor de *bits* e cada registro na tabela é representado por um *bit* nesse vetor, sendo que o *bit* é ativado se o registro possui o valor que o respectivo vetor representa. Esse tipo de índice oferece um bom desempenho nas consultas por serem utilizadas operações binárias (E, OU, NÃO) entre os vetores de bits para identificar os registros que possuem uma determinada

Figura 3 – Tempo de acesso ao disco utilizando índice *B+tree* vs. acesso sequencial.



Fonte: Lightstone, Teorey e Nadeau (2007)

combinação de valores dos atributos indexados, porém não é facilmente expansível para novos registros e valores de atributos como os índices *B+tree* (LIGHTSTONE; TEOREY; NADEAU, 2007, p. 27).

Índices do tipo *hash* são geralmente utilizados quando os dados de uma tabela não são armazenados em disco de forma ordenada. É utilizada uma função para calcular uma localização em disco a partir do valor da chave, a qual geralmente é criada com atributos que possuem valores únicos. A localização retornada pela função de *hash* identifica o endereço de um bloco inicial, e é utilizado durante as operações de atualização e consulta de dados. Este tipo de índice é recomendado para consultas que acessam um único registro da tabela buscando pelo valor da chave. Entretanto, os índices do tipo *B+tree* oferecem um desempenho equivalente, e, portanto, os índices *hash* são suportados em poucos SGBDs (LIGHTSTONE; TEOREY; NADEAU, 2007, p. 56).

3.2 Outras otimizações

Além do uso de índices, os SGBDs relacionais também oferecem outras estruturas que podem ser utilizadas na definição do modelo físico do banco de dados com o objetivo de reduzir o tempo necessário para acessar os dados. Algumas dessas funcionalidades são o particionamento, as visões materializadas e o agrupamento multidimensional, entre outras menos comuns.

O **particionamento** é uma técnica de otimização que consiste em dividir grandes volumes de dados em tabelas menores. Existem dois principais métodos para implementar o particionamento: horizontal e vertical. No particionamento horizontal, a estrutura da tabela permanece inalterada, mas os registros são armazenados em arquivos de dados separados. Essa divisão dos dados é feita a partir de algum critério definido previamente pelo DBA. Nesse tipo de particionamento, os dados são manipulados pelo SGBD como se estivessem em tabelas diferentes, com um número menor de registros por tabela, mas para a aplicação que acessa o banco de dados eles continuam sendo apresentados como uma tabela única.

Outra forma de particionamento é o vertical, onde alguns atributos menos utilizados de uma tabela são movidos para outra e apenas uma relação entre os registros é mantida. Assim, o tamanho de cada registro da tabela principal é reduzido, permitindo que o SGBD possa carregar mais registros para a memória simultaneamente.

As visões são uma técnica utilizada em bancos de dados para armazenar a definição de uma consulta e permitir que os dados dessa visão sejam acessados posteriormente sem a necessidade de saber como a consulta original foi escrita. As **visões materializadas** incrementam essa funcionalidade, armazenando o resultado da execução da consulta que forma a visão para que este seja utilizado em consultas subsequentes. Portanto, essa estratégia é útil quando uma determinada visão é acessada frequentemente e não é necessário que o resultado da consulta seja atualizado em todos os acessos.

O agrupamento (*clustering*) simples permite organizar os dados de uma tabela, armazenando os registros que possuem valores de atributos iguais no mesmo bloco em disco. A técnica de **agrupamento multidimensional** – *Multidimensional Clustering* (MDC) – estende o agrupamento simples, permitindo que dentro de um bloco os dados sejam novamente agrupados pelo do valor de outro atributo. Dessa forma, ao executar uma consulta que filtra por um ou mais atributos que fazem parte do agrupamento definido para a tabela, o SGBD pode acessar diretamente o bloco no disco que contém as informações desejadas.

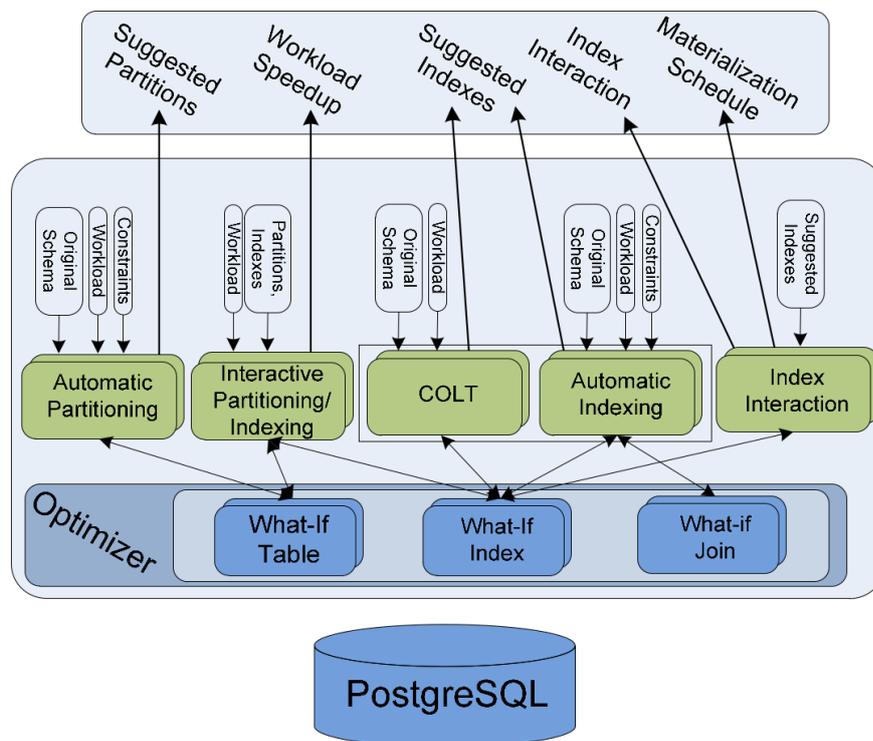
4 TRABALHOS RELACIONADOS

Este capítulo apresenta alguns dos trabalhos relacionados ao tema estudado referente ao desenvolvimento de uma ferramenta para sugestão de melhorias de performance em um banco de dados.

O trabalho de **Chaudhuri e Narasayya (1997)** apresentou a proposta de simular a existência de índices em um banco de dados e utilizar o otimizador do SGBD para estimar o custo de execução das consultas. Esse mecanismo para simulação de índices no banco de dados foi chamado de *what-if* (e-se), e desde então foi utilizado em diversos outros trabalhos. Ainda, como constatado por Lightstone, Teorey e Nadeau (2007), a proposta de Chaudhuri e Narasayya pode ser interpretada como o uso do otimizador do banco de dados no lugar de uma função de avaliação, utilizado em técnicas de busca em Inteligência Artificial.

No trabalho de **Alagiannis et al. (2010)** foi desenvolvida uma ferramenta capaz de recomendar a criação de índices e particionamento de tabelas para bancos de dados PostgreSQL. A solução apresenta uma versão modificada do otimizador de consultas do PostgreSQL, adicionando um componente *what-if* para simular a presença de índices e partições, além de outras ferramentas e módulos que executam isoladamente, como mostra a figura 4.

Figura 4 – Arquitetura do sistema desenvolvido no trabalho de Alagiannis et al.

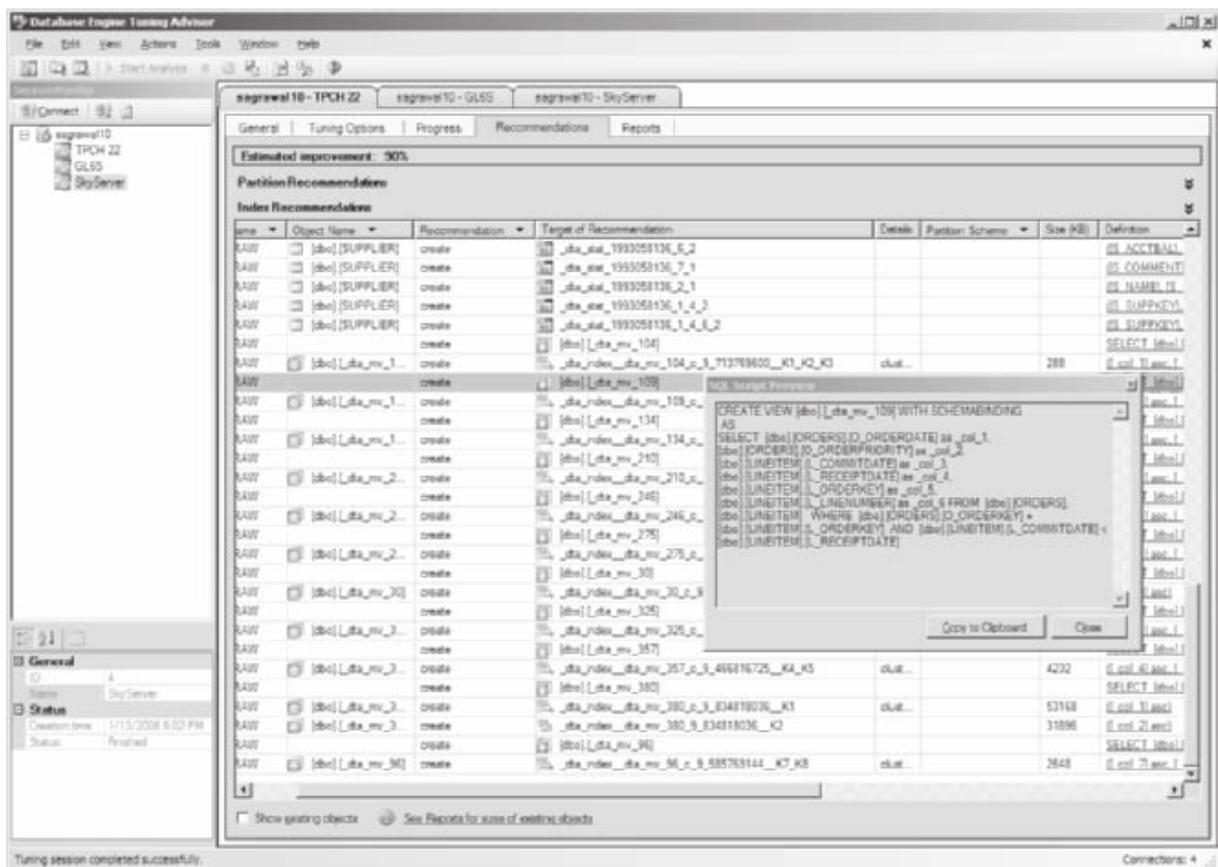


Fonte: Alagiannis et al. (2010)

Para realizar recomendações de índices, a ferramenta possibilita duas alternativas: 1) sugestão automática de índices, utilizando uma abordagem *offline*. A ferramenta utiliza o componente CoPhy¹ para sugerir os índices candidatos e avalia o custo das consultas utilizando o modelo INUM². A outra opção para recomendação de índices é a 2) otimização contínua, que utiliza uma abordagem *online* para analisar a carga de trabalho e sugerir modificações nos índices para melhorar a performance. Esta opção é desenvolvida utilizando o *framework* COLT³, porém as alterações no banco de dados não são realizadas automaticamente, apenas é retornado um novo conjunto de índices recomendados.

O trabalho de Agrawal et al. (2004) descreve a ferramenta de otimização de banco de dados integrada com o Microsoft SQL Server, o *Database Tuning Advisor* (DTA), apresentada na figura 5. As características que são suportadas por esta ferramenta para análise e recomendação de melhorias são índices, particionamento horizontal e materialização de dados.

Figura 5 – Interface da ferramenta Microsoft SQL Server DTA.



Fonte: Lightstone, Teorey e Nadeau (2007)

¹DASH; POLYZOTIS; AILAMAKI, 2011

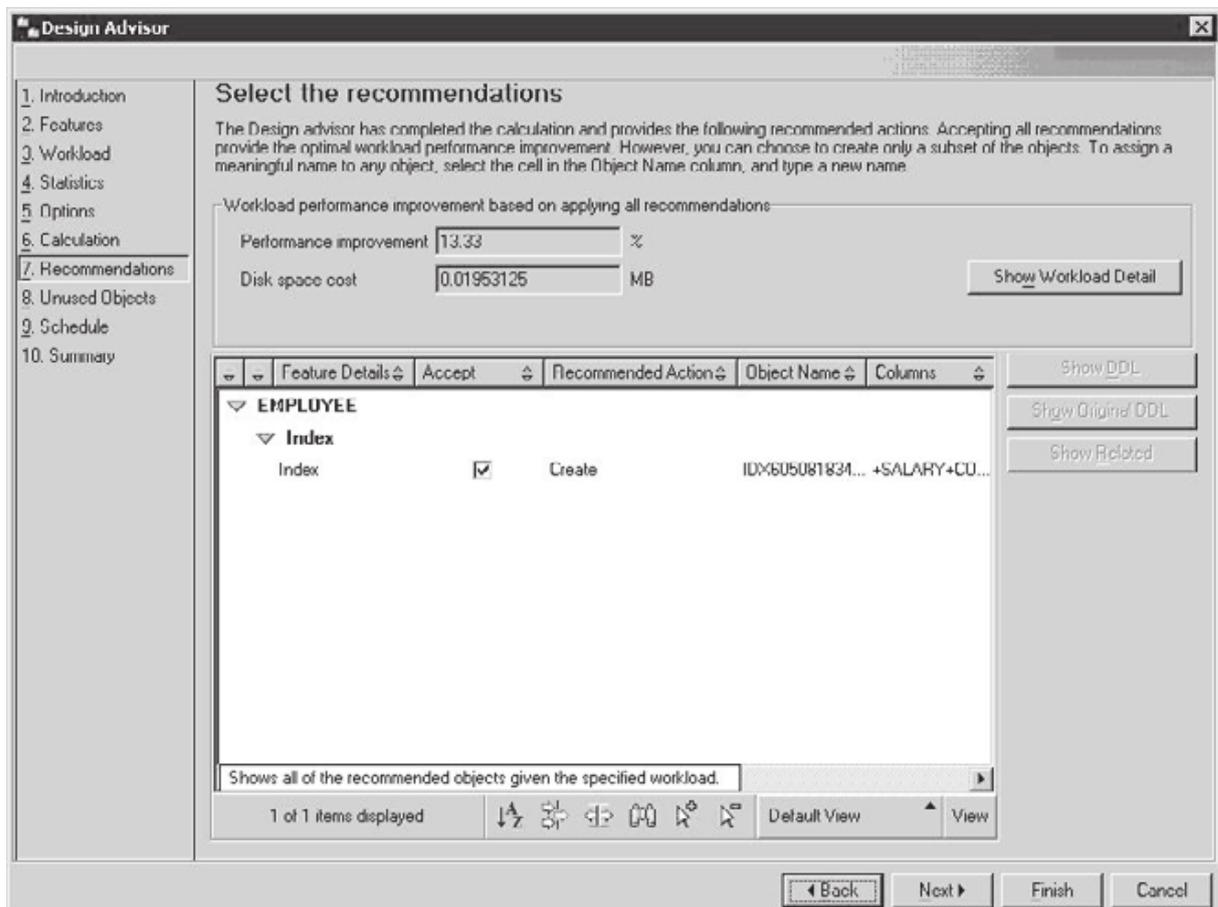
²PAPADOMANOLAKIS; DASH; AILAMAKI, 2007

³SCHNAITTER et al., 2006

Para realizar a análise e sugestão de índices, a ferramenta utiliza um componente *what-if* nativo do SQL Server, desenvolvido no trabalho de Chaudhuri e Narasayya (1998). A ferramenta utiliza uma interface com o otimizador de consultas do SGBD para calcular a estimativa de custo das consultas utilizando os índices sugeridos. Os resultados dos experimentos apresentados por Agrawal et al. (2004) mostram que as sugestões do DTA são superiores às otimizações realizadas manualmente em diferentes cenários, considerando o custo total da carga de trabalho analisada como parâmetro de comparação.

No trabalho de **Zilio et al. (2004)** é apresentada a ferramenta *DB2 Design Advisor* (figura 6), desenvolvida para o DB2 Universal Database (DB2 UDB), que oferece a sugestão de um modelo físico composto de índices, partições, visões materializadas e agrupamento multi-dimensional. O trabalho avalia duas abordagens para resolver o problema: 1) a iterativa, onde cada funcionalidade é sugerida isoladamente, porém pode ocasionar a sugestão de estruturas desnecessárias, e 2) a integrada, onde são avaliadas todas as combinações de funcionalidades e a dependência entre elas para realizar as sugestões.

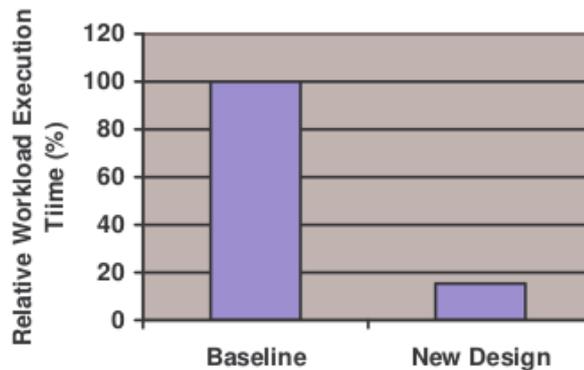
Figura 6 – Interface da ferramenta *DB2 Design Advisor*.



Fonte: Lightstone, Teorey e Nadeau (2007)

A abordagem escolhida por Zilio et al. para implementar em seu trabalho foi uma abordagem híbrida, reduzindo o espaço de busca por soluções ao procurar por estruturas que dependem de outras após ter estas definidas. Para calcular a estimativa de custo das consultas foi utilizado um componente *what-if* integrado ao otimizador do DB2. Ao final da execução, a ferramenta exibe uma lista de índices sugeridos e a opção para criá-los no banco de dados. Os resultados obtidos com a ferramenta representaram uma redução do tempo de execução de consultas entre 77% e 93% para diferentes casos. A figura 7 apresenta os resultados obtidos utilizando um banco de dados derivado de um *benchmark* TPC-H⁴, que consiste em um grande volume de dados e um conjunto de consultas complexas, criado especificamente para avaliações de desempenho.

Figura 7 – Resultados obtidos por Zilio et al. utilizando um banco de dados TPC-H.



Fonte: Zilio et al. (2004)

O **Percona Toolkit**⁵ é um conjunto de ferramentas para auxiliar na manutenção e administração de bancos de dados MySQL. Algumas das ferramentas relacionadas a performance são: identificação de índices duplicados e não utilizados, monitoramento de performance das operações de entrada/saída, análise de consultas lentas, verificação de configurações do SGBD, entre várias outras ferramentas para controle de replicação e *backup* (SCHWARTZ; NICHTER; CIZMICH, 2015).

4.1 Comparativo entre trabalhos relacionados

A tabela 1 apresenta um quadro comparativo entre os trabalhos relacionados estudados, demonstrando qual o SGBD suportado, o objetivo do trabalho, a abordagem utilizada na ferramenta e qual o modelo para avaliação do custo das consultas que foi aplicado.

⁴TPC Benchmark H: <<http://www.tpc.org/tpch/default.asp>>

⁵Percona Toolkit: <<http://www.percona.com/software/mysql-tools/percona-toolkit>>

Tabela 1 – Comparativo de trabalhos relacionados

Trabalho	SGBD	Objetivo	Abordagem	Avaliação de custo
Alagiannis et al. (2010)	PostgreSQL	Sugestão de índices e particionamento	<i>Offline e on-line</i>	Uso de componente <i>what-if</i> , acessando otimizador do SGBD para avaliar custo da consulta com <i>cache</i> dos resultados
Agrawal et al. (2004)	SQL Server	Sugestão de índices, particionamento e materialização de dados	<i>Offline</i>	Uso de componente <i>what-if</i> , acessando otimizador do SGBD para avaliar custo da consulta
Zilio et al. (2004)	DB2	Sugestão de índices, particionamento, materialização de dados e agrupamento multidimensional	<i>Offline</i>	Uso de componente <i>what-if</i> , acessando otimizador do SGBD para avaliar custo da consulta
Percona Toolkit	MySQL	Análise de configurações, identificação de índices duplicados e não utilizados	Não se aplica	Não se aplica

Fonte: Elaborado pelo autor.

5 SOLUÇÃO DESENVOLVIDA – MIST (*MYSQL INDEX SUGGESTION TOOL*)

A ferramenta MIST (*MySQL Index Suggestion Tool*, Ferramenta de Sugestão de Índices para MySQL) que foi desenvolvida neste trabalho realiza a análise da estrutura de um banco de dados e de um lista de consultas elaboradas para serem executadas nesse mesmo banco, produzindo um conjunto de índices recomendados que podem ser utilizados para otimizar a performance geral do banco de dados.

5.1 Modificação da solução proposta

A proposta de solução original deste trabalho estabeleceu que seriam realizadas modificações no código-fonte do SGBD MySQL a fim de permitir a simulação de índices sem a necessidade de criar sua estrutura completa no banco de dados. Esta decisão em fase de projeto, leia-se TC I, foi dada com base em soluções semelhantes já desenvolvidas para outros SGBDs do mercado (SQL Server, Oracle, DB2, entre outros). A partir desta simulação, pretendia-se obter as performances de processamento com os índices sugeridos pelos algoritmos implementados, utilizando os resultados destes testes para gerar a recomendação final da ferramenta.

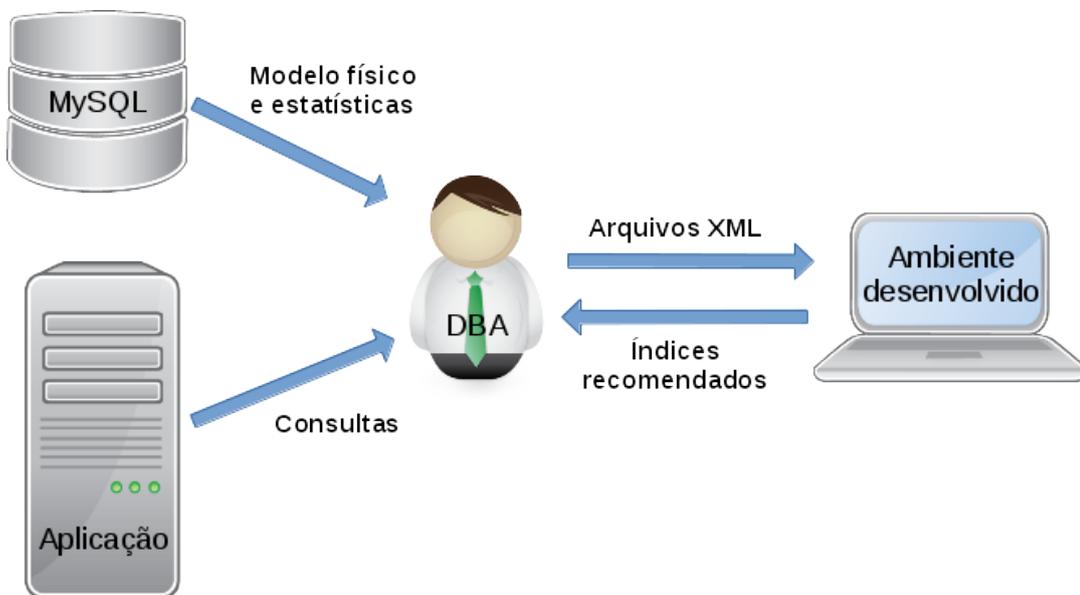
Depois de uma longa dedicação para conhecer o código-fonte do MySQL, principalmente no que tange à seleção de índices para execução de consultas e seus cálculos de performance, não foi obtido êxito nesta tarefa. Neste período, observou-se que os trabalhos relacionados são frutos de longas pesquisas, fato que não havia sido estimado. Desta forma, resolveu-se modificar o projeto de solução deste trabalho, como descrito na sequência deste capítulo.

As seções a seguir detalham as etapas de desenvolvimento e execução da ferramenta. Na seção 5.2 é apresentada de forma sucinta a arquitetura interna do ambiente desenvolvido; na seção 5.3 é descrita a forma em que os dados de entrada devem ser apresentados à ferramenta para análise; em 5.4 é apresentado o algoritmo que foi implementado para gerar um conjunto de possíveis índices para as consultas examinadas; a seção 5.5 descreve o método que foi utilizado para estimar o custo das consultas com os índices candidatos gerados na etapa anterior; a seção 5.6 apresenta o procedimento realizado para se obter a solução final com a escolha dos índices que apresentam o menor custo total para o banco de dados; por fim, a seção 5.7 apresenta os testes que foram realizados para validar a ferramenta desenvolvida, bem como os resultados obtidos.

5.2 Arquitetura interna

O MIST foi desenvolvido para ser utilizado por um administrador do banco de dados que se deseja analisar. O DBA deve coletar as consultas SQL executadas pela aplicação, o modelo físico do banco de dados e as estatísticas necessárias sobre os dados, convertendo estas informações para arquivos XML utilizados como entrada de dados para a ferramenta, como ilustrado na figura 8. Como resultado, o ambiente irá apresentar um conjunto de índices recomendados, os quais devem ser avaliados e implementados a critério do DBA.

Figura 8 – Utilização do ambiente desenvolvido.

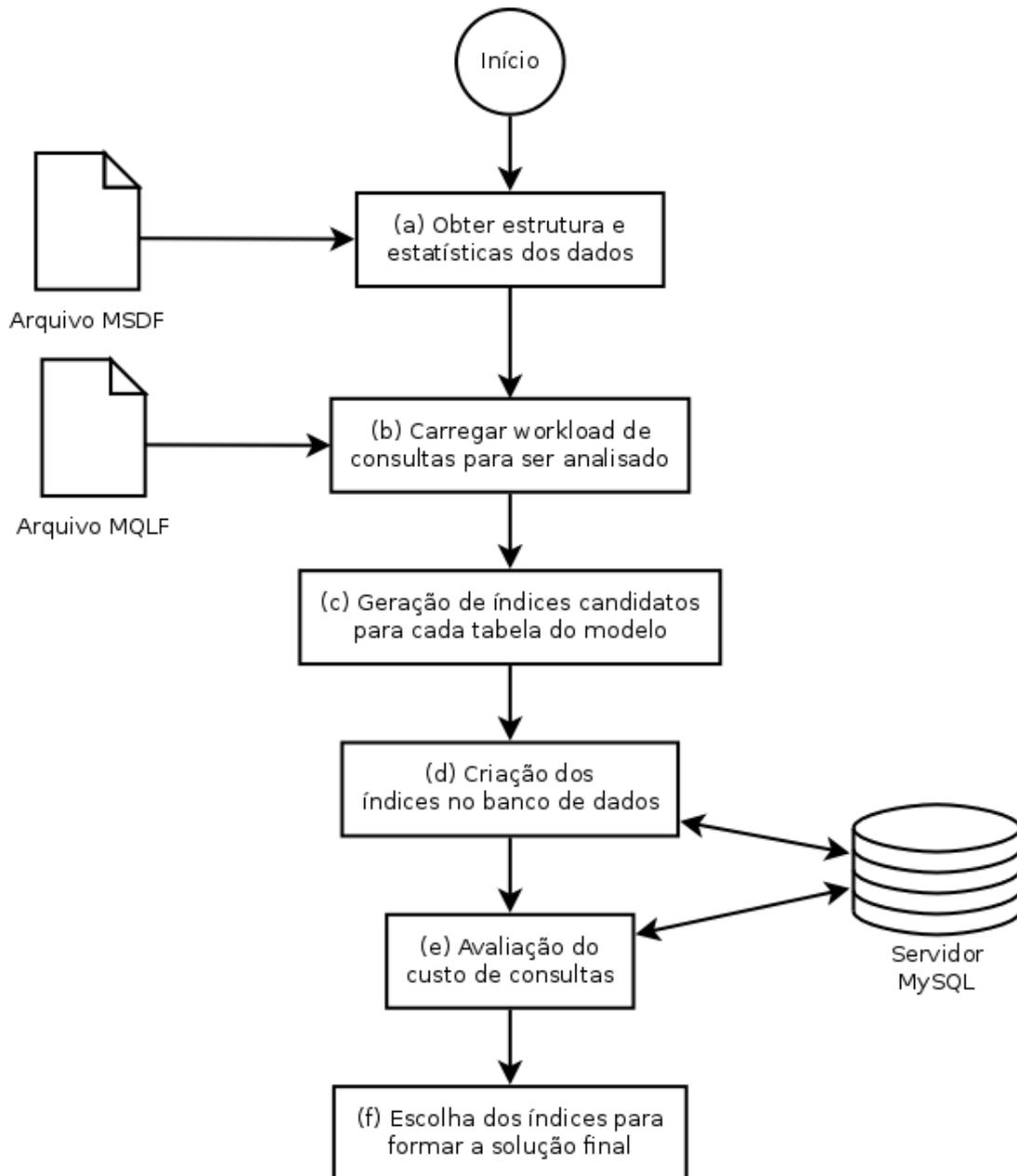


Fonte: Elaborado pelo autor.

O uso da ferramenta é formado por uma série de operações executadas sequencialmente, conforme apresentado na figura 9. Na primeira etapa, identificada na figura como (a), é feita a entrada da estrutura das tabelas e as estatísticas dos dados através de um arquivo no formato MSDF (*MIST Schema Definition File*, Arquivo de Definição do Modelo do Banco de Dados do MIST), apresentado na seção 5.3.1.

Na etapa (b), o *workload* que deve ser analisado é informado através de um arquivo no formato MQLF (*MIST Query Log File*, Arquivo de Histórico de Consultas do MIST), descrito na seção 5.3.2. Em seguida, na fase (c) são gerados os índices candidatos para cada consulta analisada, baseado em convenções básicas e recomendações do MySQL. Esta etapa é detalhada na seção 5.4. Posteriormente, em (d) os índices gerados são criados em um banco de dados e o custo das consultas com cada índice é verificado em (e) através de chamadas EXPLAIN

Figura 9 – Arquitetura interna do ambiente desenvolvido.



Fonte: Elaborado pelo autor.

para o otimizador do MySQL. Estas duas etapas são repetidas para todos os índices candidatos gerados, conforme apresentado na seção 5.5.

A última etapa (f) da ferramenta consiste em formar o conjunto de índices candidatos que será promovido como recomendação final. Este processo é descrito na seção 5.6.

5.3 Entrada de dados

Os dados que são utilizados como entrada para a ferramenta desenvolvida são a estrutura das tabelas, juntamente com algumas estatísticas sobre os dados, e um conjunto de consultas

que são executadas por alguma aplicação sobre essas tabelas, igualmente acompanhadas de algumas estatísticas disponíveis.

Para que essas informações sejam apresentadas à ferramenta, foram desenvolvidos dois formatos de arquivos baseados no padrão *eXtensible Markup Language* (XML). O primeiro deles é o MSDF (*MIST Schema Definition File*, Arquivo de Definição do Modelo do Banco de Dados do MIST), utilizado para descrever a estrutura das tabelas e índices existentes no banco de dados. O outro formato de arquivo é o MQLF (*MIST Query Log File*, Arquivo de Histórico de Consultas do MIST), que descreve um conjunto de consultas para ser analisado.

5.3.1 O formato do Arquivo de Definição do Modelo do Banco de Dados (MSDF)

O formato de arquivo MSDF possui uma estrutura como a apresentada no Código 1. O elemento raiz da estrutura possui o nome `schema` e um atributo `version` que identifica a versão do formato do arquivo. A versão desenvolvida neste trabalho é a 1.0, e futuras modificações no formato do arquivo devem incrementar esse valor, de forma que o *software* possa manter compatibilidade com diferentes versões do arquivo.

Código 1 – Exemplo de arquivo no formato MSDF.

```
<?xml version="1.0" encoding="utf-8" ?>
<schema version="1.0">
  <table name="nome_da_tabela" row-count="50000">
    <column name="id"
      type="integer"
      nullable="false"
      distinct-values="50000"
      null-values="0"/>

    <column name="coluna_int"
      type="integer"
      nullable="true"
      distinct-values="25"
      null-values="3584"/>

    <column name="coluna_texto"
      type="text"
      length="60"
      nullable="true"
      distinct-values="32054"
      null-values="264"/>

    <primary-key>
      <column name="id"/>
    </primary-key>
    <foreign-key table="outra_tabela">
      <column name="coluna_int" referenced="id"/>
    </foreign-key>
  </table>
</schema>
```

```

        </foreign-key>
    </table>
</schema>

```

Os elementos filhos do `schema` possuem o nome `table`, representando a estrutura de uma tabela existente do banco de dados. Os atributos deste elemento são `name`, cujo valor deve ser o nome da tabela, e `row-count`, que deve informar a quantidade total de registros existentes na tabela. Este e outros valores são utilizados no cálculo de algumas estatísticas sobre os dados no momento da geração de índices candidatos.

Cada coluna de uma tabela é representada no arquivo através de uma *tag* `column` vazia (contendo apenas atributos) dentro da respectiva *tag* `table`. O atributo `name` representa o nome da coluna; o atributo `type` indica qual é o tipo de dados armazenado na coluna, e deve ser um dos seguintes valores suportados: `varchar`, `char`, `text`, `tinyint`, `smallint`, `integer`, `mediumint`, `bigint`, `boolean`, `decimal`, `date`, `time`, `datetime`, `float`, `double` e `blob`. O atributo `nullable` deve conter um valor lógico `true` ou `false` indicando se a coluna aceita valores `NULL` ou não. O atributo `distinct-values` deve apresentar a quantidade de valores diferentes existentes nessa coluna entre todos os registros da tabela, desconsiderando os valores `NULL`. Estes devem ser contabilizados no atributo `null-values`, informando quantos registros possuem o valor `NULL` para essa coluna. Por fim, os atributos opcionais `length` e `precision` são utilizados apenas quando o tipo da coluna requerer essas informações.

Logo após a definição das colunas da tabela devem ser incluídas as definições de chave primária e chaves estrangeiras, quando estas existirem na tabela. Para a chave primária utiliza-se o elemento `primary-key`, sendo que cada coluna presente na chave primária é indicada através da inclusão de um elemento filho com a *tag* `column` contendo o atributo `name`. As chaves estrangeiras são definidas pela inclusão de elementos `foreign-key`, sendo um para cada chave estrangeira presente na tabela. A *tag* `foreign-key` possui o atributo `table` que deve indicar o nome da tabela referenciada. As colunas da chave são indicadas por elementos `column` filhos, contendo os atributos `name`, com o nome da coluna na tabela atual, e `referenced`, com o nome da coluna na tabela referenciada.

5.3.2 O formato do Arquivo de Histórico de Consultas (MQLF)

O arquivo de histórico de consultas (MQLF) apresenta um formato semelhante ao de definição do modelo do banco de dados (MSDF), conforme exemplificado no Código 2. O

elemento raiz da estrutura do XML é queries, que possui o atributo version com valor 1.0.

Código 2 – Exemplo de arquivo no formato MQLF.

```
<?xml version="1.0" encoding="utf-8" ?>
<queries version="1.0">
  <query id="1" count="42">
    <sql>
      SELECT
        t.id,
        t.coluna_texto,
        ot.coluna_int,
        ot.data_compra,
        ot.status,
        SUM(ot.valor)
      FROM
        nome_da_tabela t
      INNER JOIN
        outra_tabela ot
          ON ot.coluna_int = t.id
      WHERE
        ot.data_compra BETWEEN '2016-01-01' AND '2016-01-31'
        AND ot.status IN ('p', 'a', 'c')
        AND t.coluna_texto LIKE '%pesquisa%'
      GROUP BY
        t.id
      ORDER BY
        t.coluna_texto ASC,
        ot.data_compra DESC
    </sql>
    <from>
      <table>nome_da_tabela</table>
    </from>
    <joins>
      <join type="inner" table="outra_tabela">
        <condition field="coluna_int"
          type="match"/>
      </join>
    </joins>
    <where>
      <condition table="outra_tabela"
        field="data_compra"
        type="range"/>

      <condition table="outra_tabela"
        field="status"
        type="list"/>

      <condition table="nome_da_tabela"
        field="coluna_texto"
        type="like"/>
    </where>
    <groupby>
```

```

        <field table="nome_da_tabela"
            field="id"/>
    </groupby>
    <orderby>
        <field table="nome_da_tabela"
            field="coluna_texto"
            dir="asc"/>

        <field table="outra_tabela"
            field="data_compra"
            dir="desc"/>
    </orderby>
</query>
</queries>

```

Os elementos filhos do nodo raiz são identificados pela *tag* `query`, cada um representando uma consulta distinta executada pelo sistema analisado. Uma consulta é considerada distinta, no contexto deste trabalho, quando não houver outra consulta que apresente, simultaneamente, as mesmas tabelas na cláusula `FROM`, as mesmas tabelas e condições de junção nas cláusulas `JOIN`, os mesmos operadores de comparação para os mesmos campos na cláusula `WHERE`, e os mesmos campos para agrupamento e para ordenação.

O elemento `query` apresenta ainda os atributos `id`, que deve ser um número inteiro para identificar exclusivamente cada consulta dentro do mesmo arquivo, e `count`, que deve ser um número inteiro positivo indicando o número de vezes que essa determinada consulta foi executada dentro de um período observado. O valor deste atributo não necessariamente deve ser exato, mas para obter os melhores resultados a proporção de distribuição do número de execução entre todas as consultas analisadas deve ser mantida próxima ao real.

Os nodos filhos de `query` são utilizados para descrever a consulta. O primeiro é o `sql`, que deve conter o texto da consulta original, com todos os valores dos parâmetros necessários para a sua execução no banco de dados. O elemento `from` deve conter um ou mais elementos `table` com o nome das tabelas encontradas na cláusula `FROM` da consulta.

Caso a consulta realize `JOINS` com uma ou mais tabelas, deve ser incluído o elemento `joins` para descrevê-los. Este elemento possui nodos filhos com a *tag* `join`, com um atributo `type` indicando o tipo de junção realizada (`inner`, `left` ou `right`) e o atributo `table` com o nome da tabela com a qual é realizado o `JOIN`. Os nodos filhos desse elemento refletem as condições para a junção, representadas com a *tag* `condition` e os atributos `field` com o nome do campo e `type` com o tipo de comparação utilizado: `const`, `match`, `range`, `list` ou `like`.

A *tag where* lista todas as condições de filtro utilizadas no WHERE da consulta SQL. Consiste em vários elementos *condition* com os atributos *table* e *field* identificando, respectivamente, o nome da tabela e o nome do campo que é utilizado na condição, e o atributo *type* com o tipo do operador de comparação utilizado: *const*, *match*, *range*, *list* ou *like*.

Os últimos elementos necessários para descrever a consulta são o *groupby* e *orderby*, utilizados apenas no caso de a consulta incluir estes operadores. A *tag groupby* lista os campos utilizados para agrupamento na consulta, utilizando elementos *field* com os atributos *table* e *field* identificando, respectivamente, o nome da tabela e o nome do campo. O elemento *orderby* lista as regras de ordenação utilizadas na consulta, utilizando *tags field* com os atributos *table* e *field* identificando a tabela e coluna ordenada e o atributo *dir* indicando a direção da ordenação (*asc* ou *desc*).

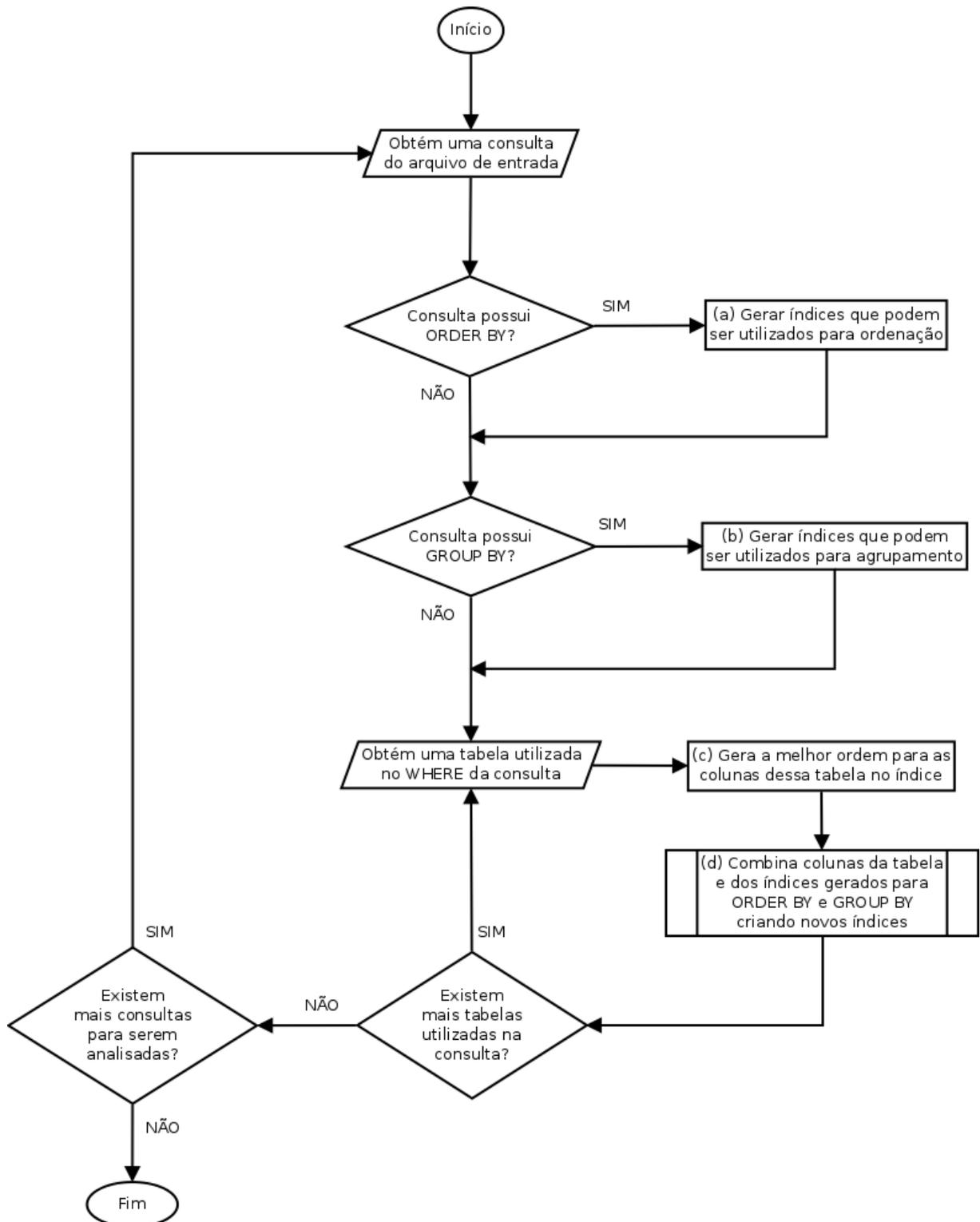
5.4 Geração de índices candidatos

A geração de índices candidatos envolve analisar as consultas que foram informadas no arquivo de entrada da ferramenta, identificando quais são os campos das tabelas que são incluídos mais frequentemente em condições de filtro, como WHERE e JOIN, bem como em outras cláusulas da consulta que afetam a forma como os dados são recuperados do banco de dados, como GROUP BY e ORDER BY.

O processo de geração dos índices candidatos deve ser otimizado de forma a não sugerir índices que ofereçam ganhos pouco significativos, visando reduzir o tempo de processamento da etapa seguinte - a estimativa de custo das consultas. Com esse objetivo, o MIST foi desenvolvido considerando as recomendações de melhores práticas na criação de índices especificamente para o banco de dados MySQL (ORACLE, 2016). Essas recomendações são utilizadas no momento de combinação de colunas para formação dos índices, influenciando na ordem em que as colunas são indexadas, como apresentado na figura 10.

Na figura 10 apresenta-se as quatro etapas intermediárias essenciais para a geração dos índices candidatos de forma otimizada, que são executadas para todas as consultas do *workload* analisado. Em (a) são identificadas as colunas incluídas na cláusula ORDER BY da consulta, limitando-se a colunas de uma mesma tabela. Em (b) um processo semelhante é realizado para as colunas da cláusula GROUP BY. A saída dessas duas etapas são os índices mais específicos que podem ser utilizados para ordenação e agrupamento dessa consulta, e são armazenados

Figura 10 – Processo de geração de índices candidatos.



Fonte: Elaborado pelo autor.

temporariamente para inclusão em uma etapa posterior.

Após as etapas (a) e (b) serem concluídas, as etapas (c) e (d) são executadas repetidamente para cada tabela diferente incluída na consulta que está sendo analisada. A etapa (c)

utiliza convenções básicas para a construção de índices, conforme descrito por Lightstone, Teorey e Nadeau (2007), como a ordem em que as colunas devem ser consideradas para inclusão no índice. Dessa forma, a implementação do MIST considera uma sequência adequada para escolher as colunas que formarão os índices sugeridos, conforme enumerado a seguir:

1. colunas que são utilizadas em comparação de igualdade com valores constantes (parâmetros de comparação informados no SQL) no WHERE ou no JOIN;
2. colunas que são utilizadas em comparação de igualdade com outras colunas no WHERE ou no JOIN;
3. colunas utilizadas no WHERE em condições de busca por intervalo de valores;
4. colunas utilizadas em comparação com uma lista de valores constantes (por exemplo, utilizando a construção IN do SQL);
5. colunas utilizadas com o operador LIKE;

Além do tipo de comparação realizado entre as colunas nas condições de WHERE e JOIN, outros fatores também são determinantes na decisão das colunas que serão adicionadas ao índice, a saber:

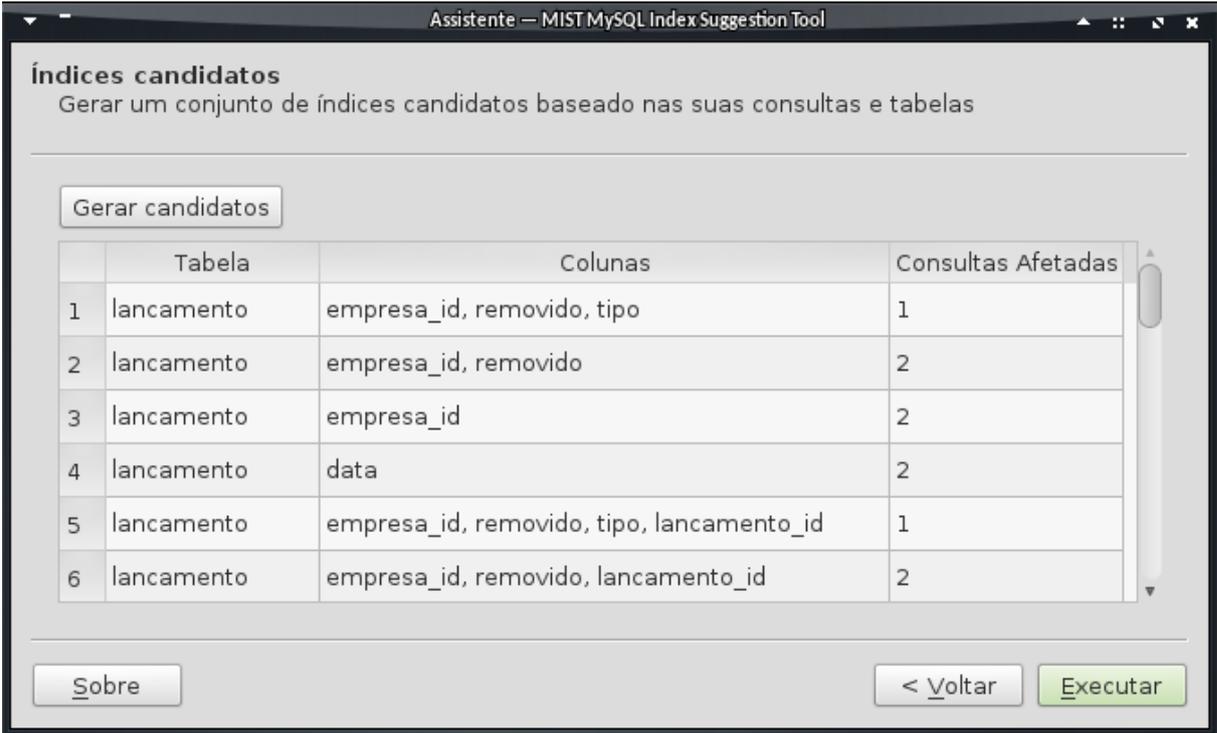
- para uma consulta que possui agrupamento e ordenação, só é possível utilizar índices para a operação de agrupamento. Assim, as colunas que estão presentes apenas na cláusula ORDER BY da consulta não são consideradas para o índice;
- índices candidatos formados por colunas que correspondem a um prefixo da chave primária da tabela são desconsiderados, uma vez que esses seriam preteridos em favor do índice primário durante a otimização da consulta;

Ainda, o MIST não considera colunas do tipo BLOB ou TEXT para serem utilizadas em índices. Isto se deve ao fato de que o MySQL não oferece suporte à indexação da coluna completa para estes tipos de dados, sendo necessário sempre informar o tamanho de um prefixo para ser indexado nessas colunas. Visando simplificar o desenvolvimento, a identificação do tamanho do prefixo ideal para estas colunas não foi implementada no MIST, impossibilitando, desta forma, que tais colunas sejam incluídas nos índices gerados.

A etapa (d) apresentada no diagrama da figura 10 é responsável por reunir os resultados das três etapas anteriores e gerar os índices candidatos. Essa geração ocorre pela combinação de prefixos dos índices de GROUP BY e ORDER BY com prefixos dos índices gerados para as condições de filtro na tabela.

A interface do MIST que exibe os índices candidatos gerados é exibida na figura 11. Os índices são exibidos no formato de uma tabela, onde cada linha corresponde a um índice candidato gerado. As duas primeiras colunas contêm a definição do índice, enquanto a terceira, "Consultas Afetadas", apresenta a quantidade de consultas que serão impactadas pelo índice, baseado nas tabelas e colunas acessadas pela consulta.

Figura 11 – Interface do MIST exibindo índices candidatos gerados.



The screenshot shows a window titled "Assistente — MIST MySQL Index Suggestion Tool". The main area is titled "Índices candidatos" with the subtitle "Gerar um conjunto de índices candidatos baseado nas suas consultas e tabelas". There is a "Gerar candidatos" button. Below it is a table with three columns: "Tabela", "Colunas", and "Consultas Afetadas". The table contains six rows of candidate indices. At the bottom, there are buttons for "Sobre", "< Voltar", and "Executar".

	Tabela	Colunas	Consultas Afetadas
1	lancamento	empresa_id, removido, tipo	1
2	lancamento	empresa_id, removido	2
3	lancamento	empresa_id	2
4	lancamento	data	2
5	lancamento	empresa_id, removido, tipo, lancamento_id	1
6	lancamento	empresa_id, removido, lancamento_id	2

Fonte: Elaborado pelo autor.

5.5 Verificação dos índices candidatos

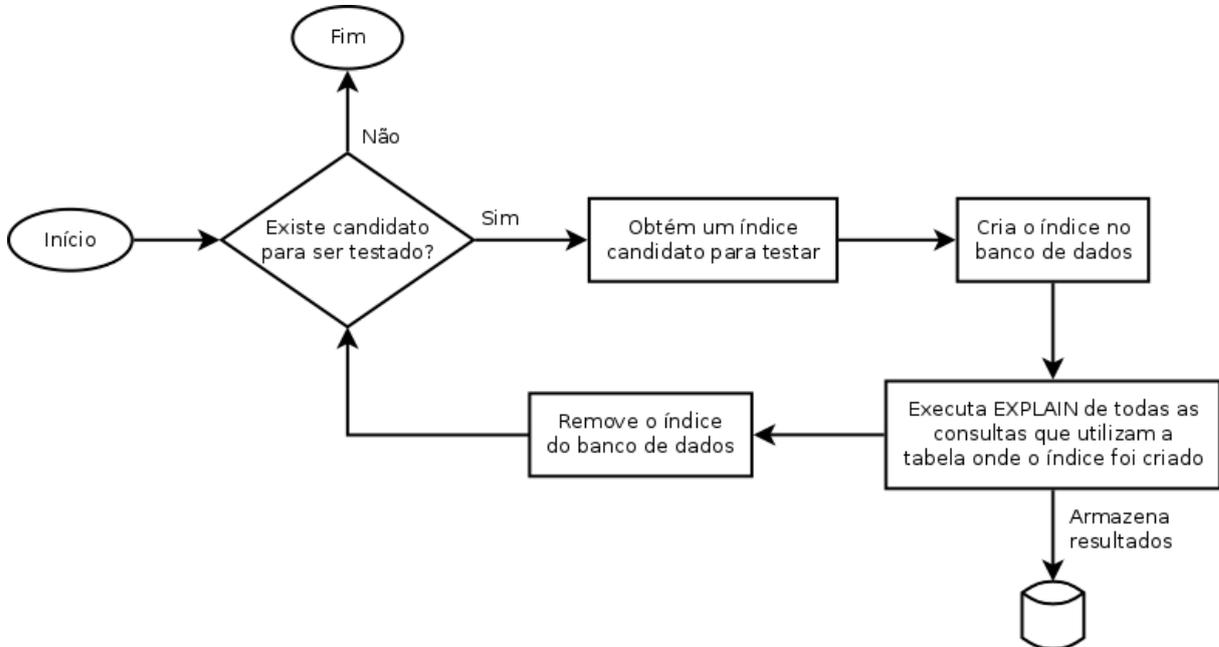
A etapa de verificação dos índices candidatos consiste em analisar o custo de execução de todas as consultas avaliadas utilizando cada índice candidato que pode afetar a sua performance. Esta análise de custo é realizada utilizando-se o comando EXPLAIN do MySQL em um banco de dados com todas as tabelas e índices candidatos criados.

O banco de dados utilizado para o processo de verificação dos índices deve ser uma cópia do banco de dados original, com todas as tabelas e dados armazenados. Apenas os índices secundários já existentes devem ser removidos, de forma a não afetar os resultados do otimizador do MySQL durante as execuções das consultas.

Cada índice candidato gerado na etapa anterior é criado no banco de dados individualmente e todas as consultas que são afetadas por ele são avaliadas. Em seguida, o índice é

removido do banco de dados e o próximo candidato é avaliado. Este processo se repete até que todos os índices candidatos tenham sido avaliados, conforme ilustrado na figura 12.

Figura 12 – Processo de verificação dos índices candidatos.



Fonte: Elaborado pelo autor.

A avaliação das consultas é realizada através de instruções EXPLAIN, que são enviadas para o servidor do banco de dados MySQL. Para ser possível obter as informações necessárias para o MIST, deve ser utilizada a versão 5.7 ou superior do servidor MySQL. Os detalhes do plano de execução da consulta são retornados pelo servidor MySQL no formato *JavaScript Object Notation* (JSON), incluindo o custo da consulta que é salvo pelo MIST vinculado ao índice utilizado e à consulta verificada.

O formato dos dados retornados no JSON depende da forma que a consulta é executada. Um exemplo do formato dessa saída é exibido no código 3. A única informação utilizada pelo MIST atualmente é a informação do custo total da consulta, encontrada na linha 5 do exemplo, que pode ser acessada pelo caminho `query_block.cost_info.query_cost`.

Código 3 – Exemplo de saída do EXPLAIN no formato JSON.

```

{
  "query_block": {
    "select_id": 1,
    "cost_info": {
      "query_cost": "1.88"
    },
    "ordering_operation": {
      "using_filesort": true,

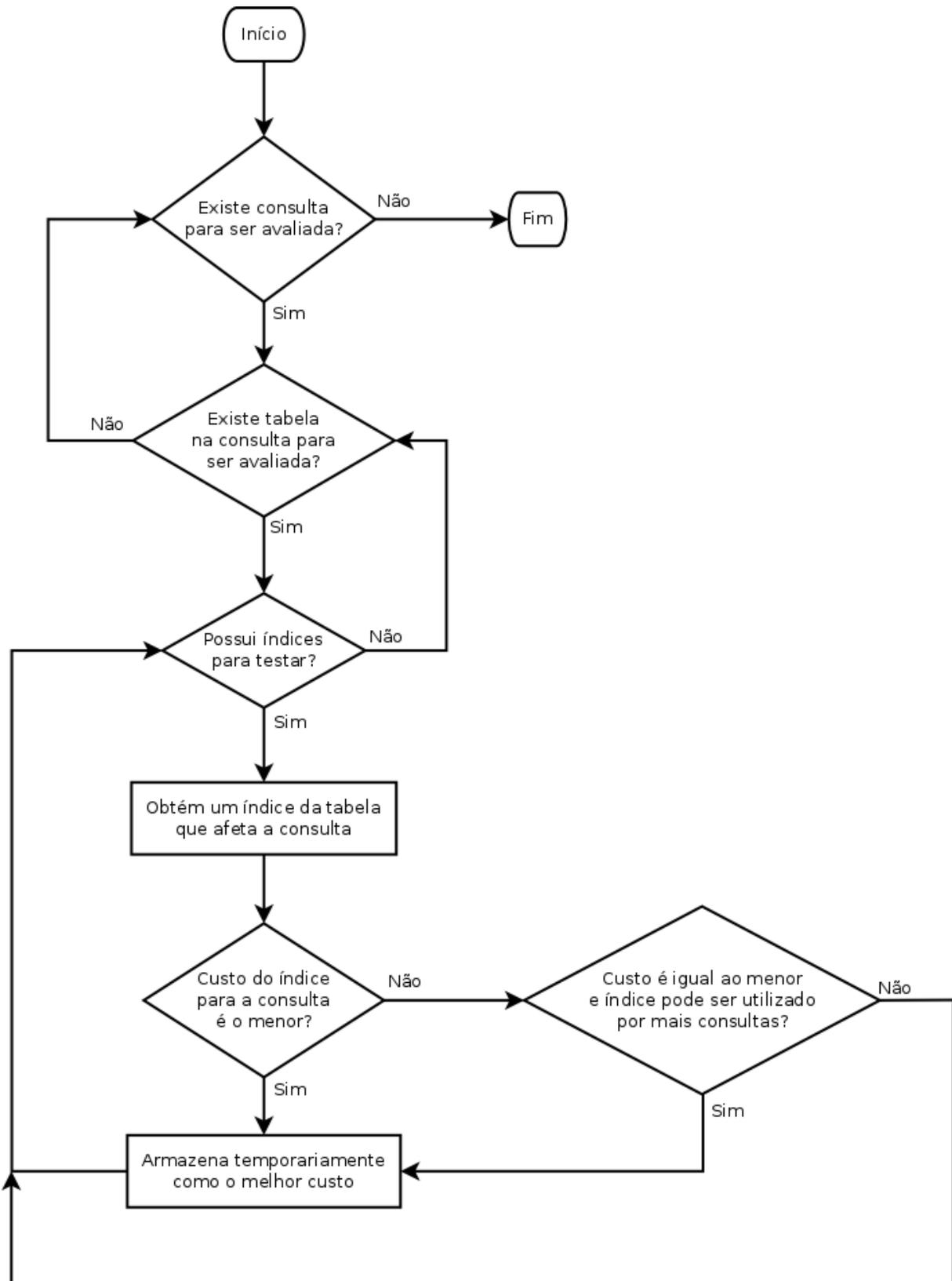
```

```

"grouping_operation": {
  "using_temporary_table": true,
  "using_filesort": false,
  "nested_loop": [
    {
      "table": {
        "table_name": "parcela",
        "access_type": "range",
        "possible_keys": [
          "__mist__index__19__"
        ],
        "key": "__mist__index__19__",
        "used_key_parts": [
          "empresa_id",
          "lancamento_id",
          "removido",
          "recorrencencia",
          "data_pagamento"
        ],
        "key_length": "18",
        "rows_examined_per_scan": 1,
        "rows_produced_per_join": 0,
        "filtered": "100.00",
        "index_condition":
          "((`mist`.`parcela`.`recorrencencia` = 0) and
            (`mist`.`parcela`.`empresa_id` = '10') and
            isnull(`mist`.`parcela`.`lancamento_id`) and
            (`mist`.`parcela`.`data_pagamento` between
              '2015-01-01' and '2016-04-18') and
            (`mist`.`parcela`.`removido` = 0))",
        "cost_info": {
          "read_cost": "1.41",
          "eval_cost": "0.03",
          "prefix_cost": "1.69",
          "data_read_per_join": "13"
        },
        "used_columns": [
          "parcela_id",
          "empresa_id",
          "lancamento_id",
          "data_pagamento",
          "recorrencencia",
          "valor",
          "categoria_id",
          "removido"
        ]
      }
    },
    {
      "table": {
        "table_name": "categoria",
        "access_type": "eq_ref",
        "possible_keys": [
          "PRIMARY"
        ]
      }
    }
  ]
}

```


Figura 13 – Fluxograma para escolha dos índices da solução final.



Fonte: Elaborado pelo autor.

O algoritmo de escolha dos índices para a solução final percorre todas as consultas que foram analisadas e, para cada tabela utilizada na consulta, escolhe o índice candidato com o menor custo. Em caso de dois ou mais índices oferecerem o mesmo custo, a ferramenta irá optar pelo índice que possa ser utilizado por mais consultas, eliminando a necessidade de índices adicionais e evitando que sejam escolhidos índices muito específicos.

5.7 Validação da ferramenta desenvolvida

Para realizar a validação da ferramenta desenvolvida, foi utilizado o banco de dados de um sistema que apresenta um grande volume de dados e executa consultas complexas frequentemente. Foram escolhidas para serem utilizadas, nesse processo de validação, sete tabelas do modelo, incluindo as que possuem a maior quantidade de registros, e quatro consultas executadas pela aplicação e que apresentam um custo de execução elevado (identificadas pelo uso de outras ferramentas de análise, como o *Percona Toolkit*¹).

Para a execução dos testes, as tabelas escolhidas foram recriadas em um servidor MySQL isolado e virtualizado em um *container* Docker, no mesmo computador utilizado para executar a ferramenta. Os dados originais foram copiados sem alterações para o novo banco de dados. Os índices secundários existentes anteriormente nas tabelas selecionadas não foram recriados, permitindo que o MIST crie apenas os índices necessários para a execução dos seus algoritmos. Apenas os índices primários das tabelas foram mantidos.

Após essas preparações iniciais, o modelo do banco de dados resultante para ser utilizado nos testes da ferramenta desenvolvida é apresentado na figura 14. O número de registros existentes em cada tabela do banco de dados é exibido na tabela 2.

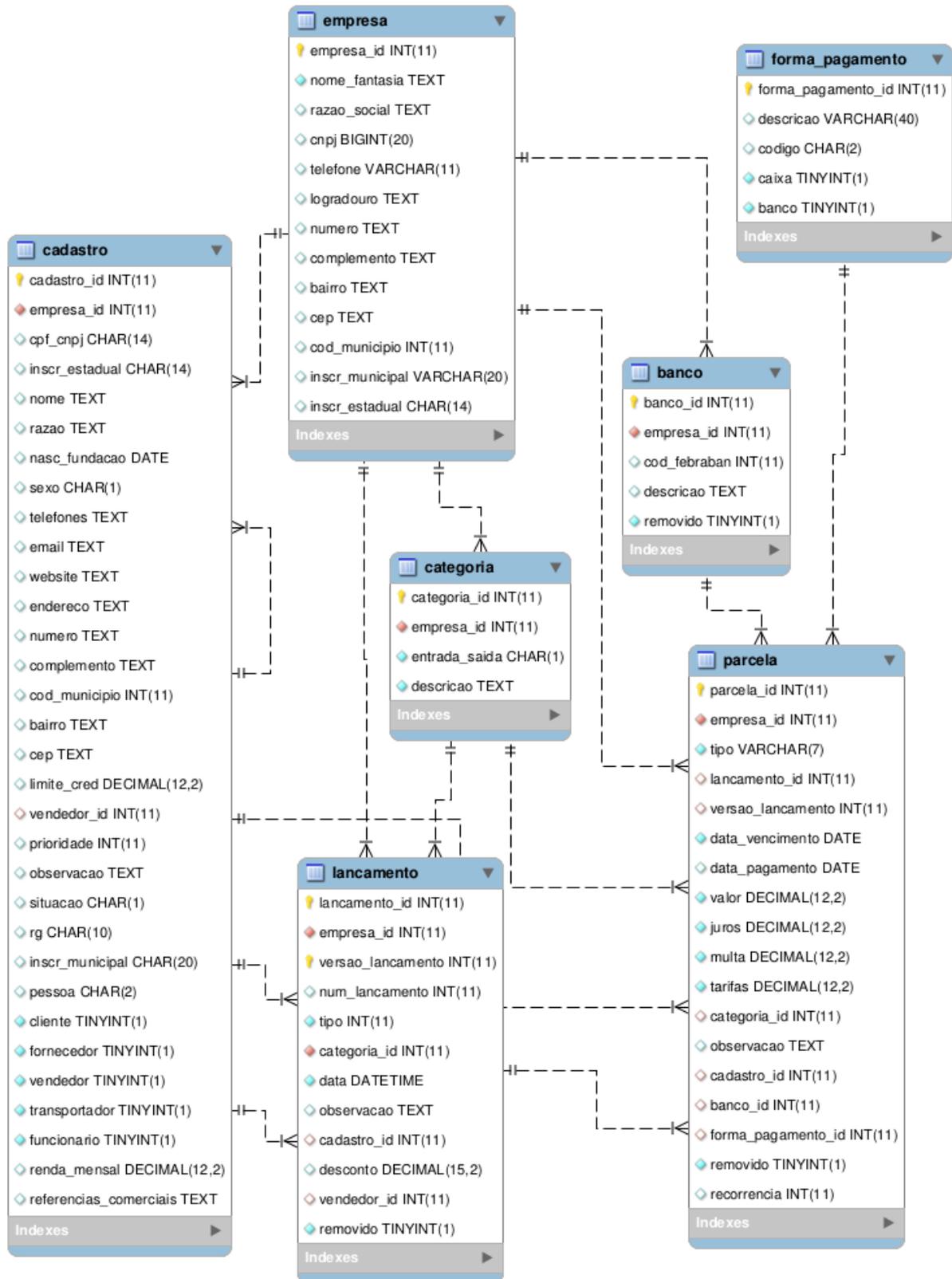
Tabela 2 – Número de registros por tabela no banco de dados de validação.

Tabela	Número de Registros
empresa	3670
forma_pagamento	13
categoria	4841
cadastro	120125
banco	2140
lancamento	748899
parcela	1159483

Fonte: Elaborado pelo autor.

¹Percona Toolkit: <<http://www.percona.com/software/mysql-tools/percona-toolkit>>

Figura 14 – Modelo do banco de dados utilizado para validação da ferramenta.



Fonte: Elaborado pelo autor.

As consultas SQL escolhidas para a realização dos testes realizam várias operações de JOIN entre as tabelas do modelo, além de operações de agrupamento por múltiplas colunas, filtro por intervalo de valores e ordenação. Uma das consultas utilizadas é apresentada para referência no código 4.

Código 4 – Exemplo de consulta SQL escolhida para os testes.

```

SELECT    parcela.parcela_id,
          parcela.tipo,
          parcela.data_pagamento,
          parcela.data_vencimento,
          parcela.observacao,
          parcela.valor,
          categoria.descricao AS categoria,
          cadastro.nome,
          cadastro.razao,
          'NE' AS status
FROM      parcela
LEFT JOIN categoria
         ON parcela.empresa_id = categoria.empresa_id
         AND parcela.categoria_id = categoria.categoria_id
LEFT JOIN cadastro
         ON cadastro.cadastro_id = parcela.cadastro_id
         AND cadastro.empresa_id = parcela.empresa_id
WHERE     parcela.empresa_id = '10'
AND       parcela.lancamento_id IS NULL
AND       parcela.data_pagamento BETWEEN '2015-01-01'
                                         AND '2016-04-18'
AND       parcela.removido = 0
AND       parcela.recorrencia = 0;

```

Ao executar a etapa de geração de índices candidatos no MIST, foram encontrados 24 possíveis índices, exibidos na figura 15. Pode-se perceber que os índices foram gerados apenas para algumas das tabelas do modelo. Considerando-se que as condições de consulta para as outras tabelas do modelo sempre referenciam a chave primária, índices secundários seriam desnecessários. Este pode ser considerado, portanto, um resultado parcial satisfatório e de acordo com o esperado.

Ao avançar para a próxima etapa, o MIST executa o processo de verificação dos índices no banco de dados, conforme descrito anteriormente na seção 5.5. Ao terminar a execução desse processo, é exibida a saída do programa, que consiste nos índices recomendados. Nesse processo de validação da ferramenta, foram selecionados cinco índices para formar o resultado, como mostra a figura 16. O MIST também exhibe, apenas para efeito de comparação, a soma do custo estimado de execução das consultas sem nenhum índice gerado no banco de dados e com todos os índices recomendados materializados.

Figura 15 – Índices candidatos gerados com o banco de dados de validação.

Assistente — MIST MySQL Index Suggestion Tool

Índices candidatos
Gerar um conjunto de índices candidatos baseado nas suas consultas e tabelas

Gerar candidatos

	Tabela	Colunas	Consultas Afetadas
1	lancamento	empresa_id, removido, tipo	1
2	lancamento	empresa_id, removido	2
3	lancamento	empresa_id	2
4	lancamento	data	2
5	lancamento	empresa_id, removido, tipo, lancamento_id	1
6	lancamento	empresa_id, removido, lancamento_id	2
7	lancamento	empresa_id, lancamento_id	2
8	lancamento	empresa_id, removido, tipo, lancamento_id, data	1
9	lancamento	empresa_id, removido, lancamento_id, data	2
10	lancamento	empresa_id, lancamento_id, data	2
11	parcela	removido, ocorrencia, lancamento_id, versao_lancamento, empresa_id	2
12	parcela	removido, ocorrencia, lancamento_id, versao_lancamento	2
13	parcela	removido, ocorrencia, lancamento_id	2
14	parcela	removido, ocorrencia	2
15	parcela	removido	2
16	categoria	empresa_id, categoria_id	4
17	categoria	empresa_id	4
18	cadastro	empresa_id, cadastro_id	3
19	cadastro	empresa_id	3
20	parcela	empresa_id, lancamento_id, removido, ocorrencia, data_pagamento	2
21	parcela	empresa_id, lancamento_id, removido, ocorrencia	2
22	parcela	empresa_id, lancamento_id, removido	2
23	parcela	empresa_id, lancamento_id	2
24	parcela	empresa_id	2

Sobre < Voltar Executar

Fonte: Elaborado pelo autor.

O custo total estimado para a execução das quatro consultas analisadas quando não existe nenhum índice criado no banco de dados foi de 1.040.877,3, enquanto que o custo utilizando-se os índices sugeridos foi de 44.166,68. Isso representa uma redução do custo de aproximadamente 95,8%. Essa comparação, no entanto, não apresenta uma otimização real da performance

Figura 16 – Índices recomendados pelo MIST para o banco de dados de validação.

Custo total sem índices:		Custo total com os índices sugeridos:	
1040877.3		44166.68	
Índices sugeridos			
	Tabela	Colunas	Consultas Afetadas
1	cadastro	empresa_id, cadastro_id	3
2	lancamento	empresa_id, removido	2
3	categoria	empresa_id, categoria_id	4
4	parcela	empresa_id, lancamento_id, removido, ocorrencia, data_pagamento	2
5	parcela	removido, ocorrencia, lancamento_id	2

Fonte: Elaborado pelo autor.

do sistema, visto que um banco de dados sem nenhum índice raramente é utilizado em aplicações reais no mercado.

Para se obter um resultado imparcial, seria necessário realizar a comparação do custo com um conjunto de índices previamente selecionado pelo administrador do banco de dados, como, por exemplo, os índices já existentes no banco de dados original. Porém, por uma casualidade, todos os índices recomendados pelo MIST já são utilizados pelo sistema analisado. Desta forma, não é possível determinar conclusivamente se os índices que foram recomendados apresentam o melhor desempenho possível. Todavia, é possível concluir que a solução oferecida pela ferramenta é válida, visto que um DBA obteve o mesmo conjunto de índices anteriormente, embora utilizando-se de processos de verificação manuais.

6 CONCLUSÃO

A otimização da performance de bancos de dados é de vital importância para a vasta maioria dos sistemas existentes no mercado que utilizam o modelo de dados relacional. Dado o rápido crescimento da quantidade de dados a ser armazenada por estes sistemas, o desempenho destes deve ser aprimorado continuamente.

Para atingir o objetivo proposto neste trabalho, foi desenvolvida uma ferramenta capaz de recomendar opções de índices para serem utilizados em um banco de dados analisado, visando reduzir o custo de processamento das consultas no servidor.

Na tabela 1, apresentada no capítulo 4, foram listados alguns dos trabalhos já desenvolvidos por outros autores da área e foi realizada uma comparação entre as contribuições dos mesmos. Retornando-se a esta tabela, pode-se agora incluir as características da aplicação desenvolvida (tabela 3).

Tabela 3 – Contribuição do trabalho desenvolvido

Trabalho	SGBD	Objetivo	Abordagem	Avaliação de custo
Weiland e Kroth, 2016	MySQL	Sugestão de índices	<i>Offline</i>	Consultas ao otimizador do SGBD utilizando índices materializados

Fonte: Elaborado pelo autor.

Os resultados obtidos com a ferramenta desenvolvida neste trabalho foram satisfatórios. Porém, pode-se observar ainda que existem vários ajustes que poderiam ser desenvolvidos na busca de melhores resultados. Em comparação aos trabalhos relacionados, o MIST ainda não possui algumas funcionalidades que seriam muito úteis, porém que não foram implementadas devido ao escopo e tempo limitado do trabalho.

A primeira melhoria detectada refere-se à entradas dos dados na ferramenta. A solução desenvolvida, utilizando arquivos XML, ainda exige um processamento manual dos dados para formatá-los de acordo com o esperado pela ferramenta. Idealmente, o MIST poderia conectar-se diretamente a um banco de dados configurado para obter todas as informações essenciais sobre as tabelas e estatísticas sobre os dados, sem a necessidade de intervenção manual.

Também referente à entrada de dados, o MIST poderia ser integrado com o *parser* do MySQL de modo a interpretar as consultas a serem analisadas sem exigir uma conversão manual destas para um arquivo XML. Além disso, esta integração possibilitaria que o MIST avaliasse consultas mais complexas, incluindo união e sub-consultas. Adicionar suporte para avaliar

instruções de atualização de dados, como INSERTs e UPDATEs, também seria uma funcionalidade importante para a ferramenta.

Outras melhorias identificadas são relacionadas à otimização do programa, que pode ser obtida de duas formas: 1) utilização de melhores heurísticas para a geração de índices candidatos, de forma a gerar uma quantidade menor de índices para serem verificados no banco de dados; ou 2) implementação do suporte a simulação de índices no MySQL, removendo a necessidade da criação dos índices materializados e utilizando apenas as estatísticas calculadas sobre os dados.

REFERÊNCIAS

- AGRAWAL, S. et al. Database tuning advisor for Microsoft SQL Server 2005. In: *VLDB*. Very Large Data Bases Endowment Inc., 2004. Disponível em: <<http://research.microsoft.com/apps/pubs/default.aspx?id=76555>>. Acesso em: 12 jul. 2015.
- ALAGIANNIS, I. et al. An automated, yet interactive and portable DB designer. In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, 2010. (SIGMOD '10), p. 1183–1186. ISBN 978-1-4503-0032-2. Disponível em: <<http://doi.acm.org/10.1145/1807167.1807314>>. Acesso em: 13 jul. 2015.
- BELL, C. *Expert MySQL*. 2nd. ed. Berkely, CA, USA: Apress, 2012. ISBN 978-1-4302-4659-6.
- CHAUDHURI, S.; NARASAYYA, V. R. An efficient cost-driven index selection tool for Microsoft SQL Server. In: *Proceedings of the 23rd International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997. (VLDB '97), p. 146–155. ISBN 1-55860-470-7. Disponível em: <<http://dl.acm.org/citation.cfm?id=645923.673646>>. Acesso em: 2 ago. 2015.
- CHAUDHURI, S.; NARASAYYA, V. R. AutoAdmin "what-if" index analysis utility. *SIGMOD Rec.*, ACM, New York, NY, USA, v. 27, n. 2, p. 367–378, jun. 1998. ISSN 0163-5808. Disponível em: <<http://doi.acm.org/10.1145/276305.276337>>. Acesso em: 12 jul. 2015.
- DASH, D.; POLYZOTIS, N.; AILAMAKI, A. Cophy: A scalable, portable, and interactive index advisor for large workloads. *Proc. VLDB Endow.*, VLDB Endowment, v. 4, n. 6, p. 362–372, mar. 2011. ISSN 2150-8097. Disponível em: <<http://dx.doi.org/10.14778/1978665.1978668>>. Acesso em: 26 ago. 2015.
- HEUSER, C. A. *Projeto de banco de dados*. 6. ed. Porto Alegre, RS: Bookman, 2009. v. 4. ISBN 978-85-7780-452-8.
- LIGHTSTONE, S. S.; TEOREY, T. J.; NADEAU, T. *Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007. ISBN 978-0-12-369389-1.
- ORACLE CORPORATION. *MySQL 5.7 Reference Manual: How MySQL Uses Indexes*. [S.l.], 2016. Disponível em: <<http://dev.mysql.com/doc/refman/5.7/en/mysql-indexes.html>>.
- PAPADOMANOLAKIS, S.; DASH, D.; AILAMAKI, A. Efficient use of the query optimizer for automated physical design. In: *Proceedings of the 33rd International Conference on Very Large Data Bases*. VLDB Endowment, 2007. (VLDB '07), p. 1093–1104. ISBN 978-1-59593-649-3. Disponível em: <<http://dl.acm.org/citation.cfm?id=1325851.1325974>>. Acesso em: 26 ago. 2015.
- PETRAKI, E.; IDREOS, S.; MANEGOLD, S. Holistic indexing in main-memory column-stores. In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, 2015. (SIGMOD '15), p. 1153–1166. ISBN 978-1-4503-2758-9. Disponível em: <<http://doi.acm.org/10.1145/2723372.2723719>>. Acesso em: 2 jul. 2015.

SCHNAITTER, K. et al. COLT: Continuous on-line tuning. In: *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, 2006. (SIGMOD '06), p. 793–795. ISBN 1-59593-434-0. Disponível em: <<http://doi.acm.org/10.1145/1142473.1142592>>. Acesso em: 5 ago. 2015.

SCHWARTZ, B.; NICHTER, D.; CIZMICH, F. *Percona Toolkit Documentation*. Release 2.2.15. [S.l.], 2015. Disponível em: <<https://learn.percona.com/download-percona-toolkit-2-2-manual>>. Acesso em: 26 out. 2015.

THALHEIM, B.; TROPMANN, M. Performance forecasting for performance critical huge databases. In: *Proceedings of the 2011 Conference on Information Modelling and Knowledge Bases XXII*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2011. p. 206–225. ISBN 978-1-60750-689-8. Disponível em: <<http://dl.acm.org/citation.cfm?id=1972754.1972767>>. Acesso em: 25 jul. 2015.

ZILIO, D. C. et al. DB2 design advisor: Integrated automatic physical database design. In: *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*. VLDB Endowment, 2004. (VLDB '04), p. 1087–1097. ISBN 0-12-088469-0. Disponível em: <<http://dl.acm.org/citation.cfm?id=1316689.1316783>>. Acesso em: 2 ago. 2015.