|  |
| --- |
| UNIVERSIDADE REGIONAL DE BLUMENAU  CENTRO DE CIÊNCIAS EXATAS E NATURAIS  CURsO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO |
| NOVA ORGANIZAÇÃO PARA ESTRUTURA DE DADOS EM BANCOS RELACIONAIS: ESTUDO DE CASO  João Paulo Poffo |
| bLUMENAU  2009  2009/1-10 |

|  |
| --- |
| João Paulo Poffo  NOVA ORGANIZAÇÃO PARA ESTRUTURA DE DADOS EM BANCOS RELACIONAIS: ESTUDO DE CASO  Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.  Prof. Marcel Hugo, Orientador |
| bLUMENAU  2009  2009/1-10 |
| NOVA ORGANIZAÇÃO PARA ESTRUTURA DE DADOS EM BANCOS RELACIONAIS: ESTUDO DE CASO  Por  JOão Paulo poffo  Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Presidente: Prof. Marcel Hugo, Orientador, FURB  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Membro: Prof. Nome do professor, Titulação – FURB  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Membro: Prof. Nome do professor, Titulação – FURB |
| Blumenau, dia de mês de ano [data da apresentação] |

À Vera Lúcia que me ensinou sobre vida ao abrir meus olhos para a morte.

AGRADECIMENTOS

A minha noiva que soube identificar em seu companheiro os momentos em que ele era seu querido noivo daqueles em que ele era o escritor egoísta, chato e ranzinza desta monografia.

Aos meus amigos, por sempre me lembrarem que falta tão pouco.

Ao meu orientador, Marcel Hugo, por me dar forças me assustando com prazos que pareciam inalcançáveis.

Conhecer não é demonstrar nem explicar, é aceder à visão.

Antoine de Saint-Exupéry

RESUMO

Para elaborar o resumo é necessário concluir os testes da prova de conceito.

[Só elaborar ao terminar a monografia] RELEVANCIA DO TRABALHO - Ajustar: Os desenvolvedores, cada vez mais, procuram um equilíbrio entre redundância de informação e performance de seus sistemas. A redundância em um SGBD pode ser atingida criando índices, tabelas desnormalizadas, entre outros. A busca desse equilíbrio se dá pela incapacidade dos SGBDs fornecerem estes dois atributos simultaneamente em uma arquitetura. Existem muitas estruturas de dados aperfeiçoadas para cada área de negócio ou problema a ser resolvido. Cada SGBD implementa estas soluções sob sua ótica de domínio. Estas implementações nem sempre se aplicam a todos os casos, contudo, são muito importantes para casos específicos. Quanto maior o grau de normalização de uma estrutura de dados, maior o impacto sobre a performance do sistema o que em contrapartida é indispensável para uma organização mais próxima da realidade. Propõe-se o desenvolvimento de uma solução para este paradoxo através de uma arquitetura de banco de dados onde seja possível criar algoritmos eficientes em estruturas não redundantes. Isso pode ser atingido através da independência da necessidade de índices, tornando toda informação ordenada, o que possibilitaria uma normalização não mais acompanhada de queda de desempenho e que manteria a agilidade na recuperação da informação. O resumo é uma apresentação concisa dos pontos relevantes de um texto. Informa suficientemente ao leitor, para que este possa decidir sobre a conveniência da leitura do texto inteiro. Expõe finalidades, metodologia, resultados e conclusões. Para monografias, recomenda-se que os resumos tenham até 250 palavras. O resumo deve ser composto de uma seqüência corrente de frases concisas e não de uma enumeração de tópicos. Deve-se usar a terceira pessoa do singular e verbo na voz ativa. O resumo deve ser escrito em um único texto corrido (sem parágrafos).

Palavras-chave: MySQL. Tipo de tabela. *Storage engine*.

ABSTRACT

Para elaborar o *abstract* é necessário concluir os testes da prova de conceito.

[Traduzir o Resumo] *Abstract* é o resumo traduzido para o inglês. *Abstract* vem em uma nova folha, logo após o resumo. Escrever com letra normal (sem itálico).

Key-words: MySQL. Table type. Storage engine.

LISTA DE ilustrações

[Figura 1 - Teste Tokutek: inserção de um bilhão de registros 19](#_Toc232491084)

[Figura 2 - Teste Tokutek: inserção/exclusão de quinhentos milhões registros 20](#_Toc232491085)

[Figura 3 - Teste Tokutek: consulta em tabela com índices de árvores fractais 20](#_Toc232491086)

[Figura 4 - Arquitetura global do MySQL com *storage engines* plugáveis 21](#_Toc232491087)

[Figura 5 - Tabela com índice do tipo *Bitmap* 25](#_Toc232491088)

[Figura 6 - Estrutura comum de tabela 27](#_Toc232491089)

[Figura 7 - Nova estrutura de tabela 27](#_Toc232491090)

[Figura 9 - Diagrama de casos de uso 29](#_Toc232491091)

[Quadro 1 - Caso de uso Criar tabela 30](#_Toc232491092)

[Quadro 2 - Caso de uso Remover tabela 30](#_Toc232491093)

[Quadro 3 - Caso de uso Inserir registro 31](#_Toc232491094)

[Quadro 4 - Caso de uso Atualizar registro 31](#_Toc232491095)

[Quadro 5 - Caso de uso Remover registro 32](#_Toc232491096)

[Quadro 6 - Caso de uso Selecionar registro 32](#_Toc232491097)

[Figura 9 - Diagrama de classes do *storage engine* 33](#_Toc232491098)

[Figura 11 - Definição da classe ha\_vogal e interface handler 34](#_Toc232491099)

[Figura 12 - Definição da classe vogal\_manipulation 35](#_Toc232491100)

[Figura 13 - Definição da classe vogal\_definition 35](#_Toc232491101)

[Figura 14 - Definição da classe vogal\_storage 35](#_Toc232491102)

[Figura 15 - Definição da classe vogal\_cache 36](#_Toc232491103)

[Figura 16 - Estrutura do arquivo de dados 37](#_Toc232491104)

[Figura 17 - Estrutura de um bloco genérico 37](#_Toc232491105)

[Figura 18 - Estrutura do cabeçalho do bloco 37](#_Toc232491106)

[Figura 19 - Estrutura do corpo do bloco do Tipo de Tabela 37](#_Toc232491107)

[Figura 20 - Estrutura da árvore de registros 38](#_Toc232491108)

[Figura 21 - Estrutura da árvore de RIDs 38](#_Toc232491109)

[Quadro 7 - Estruturas de controle e tipos padrão 40](#_Toc232491110)

[Quadro 8 - Métodos de escrita de dados nos blocos 41](#_Toc232491111)

[Quadro 9 - Método de coleta dos blocos livres 41](#_Toc232491112)

[Quadro 10 - Definição do método de abertura de tabelas 42](#_Toc232491113)

[Quadro 11 - Definição do método para consulta de registros 43](#_Toc232491114)

[Quadro 12 - Definição do método para escrita de registros 44](#_Toc232491115)

[Quadro 13 - Definição do método de leitura do registro 45](#_Toc232491116)

[Quadro 14 - DDL de criação de tabela do tipo Vogal 46](#_Toc232491117)

[Quadro 15 - DML de inserção de registro 46](#_Toc232491118)

[Quadro 16 - DML de alteração de registro 46](#_Toc232491119)

[Quadro 17 - DML de exclusão de registro 46](#_Toc232491120)

[Quadro 18 - DDL de remoção de tabela 46](#_Toc232491121)

[Quadro 19 - DML de consulta de registro 46](#_Toc232491122)

[Figura 22 - Consulta e resposta do banco de dados 46](#_Toc232491123)

[Quadro 20 - Comparativo de limitações dos tipos de tabela Vogal e Maria 48](#_Toc232491124)

[Quadro 21 - Exemplo execução OSDB 51](#_Toc232491125)

[Quadro 22 - Mapeamento dos resultados esperados com as funções do OSDB 52](#_Toc232491126)

[Quadro 23 - Limitações dos tipos de tabela 53](#_Toc232491127)

[Quadro 24 - Capacidades dos tipos de tabela 53](#_Toc232491128)

[Quadro 25 - Resultados dos testes com cem mil registros 53](#_Toc232491129)

[Figura 23 - Diagrama de classes completo do Tipo de Tabela 57](#_Toc232491130)

LISTA DE SIGLAS

ACID - *Atomic, Consistency, Isolation, Durability*

API - *Application Programming Interface*

DDD - *Data Display Debugger*

DDL - *Data Definition Language*

DML - *Data Manipulation Language*

DVD - *Digital Video Disc*

EA - *Enterprise Architect*

GCC - GNU *Compiler Collection*

GDB - GNU *DeBugger*

ISAM - *Indexed Sequential Access Method*

LOB *- Large Object*

OSDB - *Open Source Database Benchmark*

RF - Requisito Funcional

RID - *Record IDentification*

RN - Regra de Negócio

RNF - Requisito Não Funcional

SGBD - Sistema Gerenciador de Banco de Dados

SQL - *Structured Query Language*

UML - *Unified Modeling Language*

SUMÁRIO

[1 Introdução 13](#_Toc232491145)

[1.1 OBJETIVOS DO TRABALHO 14](#_Toc232491146)

[1.2 estrutura do trabalho 14](#_Toc232491147)

[2 FUNDAMENTAÇÃO TEÓRICA 15](#_Toc232491148)

[2.1 ESTRUTURAS DE DADOS 15](#_Toc232491149)

[2.2 SISTEMAS GERENCIADORES DE BANCO DE DADOS 16](#_Toc232491150)

[2.2.1 Armazenamento 16](#_Toc232491151)

[2.2.2 Arquitetura de armazenamento 17](#_Toc232491152)

[2.2.3 Índices 17](#_Toc232491153)

[2.2.4 Índices Árvore B 18](#_Toc232491154)

[2.3 tipos de tabela do MySQL 20](#_Toc232491155)

[2.3.1 Arquitetura de *storage engines* do MySQL 22](#_Toc232491156)

[2.4 *BENCHMARKING* COMO Prova de conceito 22](#_Toc232491157)

[2.5 OSDB 23](#_Toc232491158)

[2.6 trabalhos correlatos 23](#_Toc232491159)

[2.6.1 Maria: tipo de tabela 24](#_Toc232491160)

[2.6.2 Indexação *Bitmap* 24](#_Toc232491161)

[2.6.3 Comparativo de desempenho entre bancos de dados de código aberto 25](#_Toc232491162)

[3 DESENVOLVIMENTO DO TIPO DE TABELA 27](#_Toc232491163)

[3.1 requisitos principais do problema a ser trabalhado 27](#_Toc232491164)

[3.2 ESPECIFICAÇÃO 28](#_Toc232491165)

[3.2.1 Diagrama de casos de uso 29](#_Toc232491166)

[3.2.2 Diagrama de classes 33](#_Toc232491167)

[3.2.3 Estrutura do arquivo de dados 37](#_Toc232491168)

[3.3 IMPLEMENTAÇÃO 39](#_Toc232491169)

[3.3.1 Técnicas e ferramentas utilizadas 39](#_Toc232491170)

[3.3.2 Código desenvolvido 39](#_Toc232491171)

[3.3.3 Operacionalidade da implementação 45](#_Toc232491172)

[3.4 RESULTADOS E DISCUSSÃO 47](#_Toc232491173)

[4 DESENVOLVIMENTO DA PROVA DE CONCEITO 49](#_Toc232491174)

[4.1 requisitos principais do problema a ser trabalhado 49](#_Toc232491175)

[4.2 ESPECIFICAÇÃO 49](#_Toc232491176)

[4.3 IMPLEMENTAÇÃO 51](#_Toc232491177)

[4.3.1 Técnicas e ferramentas utilizadas 52](#_Toc232491178)

[4.4 RESULTADOS E DISCUSSÃO 53](#_Toc232491179)

[5 CONCLUSÕES 54](#_Toc232491180)

[5.1 EXTENSÕES 54](#_Toc232491181)

# Introdução

Atualmente é irrelevante como um programa de computador trata suas informações internamente, o mais importante é o seu resultado visível. Pois, caso o consumo de memória, processador ou espaço em disco forem altos, existem a cada dia novas tecnologias, mais rápidas, poderosas, e com preços mais baixos. Entretanto, há também a necessidade de manipular mais informação em menos tempo. Neste contexto, a evolução do hardware, apesar de contundente, sozinha, é insuficiente para preencher completamente esta lacuna. De tempos em tempos têm-se a necessidade de rever antigos conceitos e analisar se são realmente as melhores alternativas para resolver os problemas de hoje.

Existem várias formas de organizar as estruturas de bancos de dados relacionais. Desde as mais primitivas como arquivos binários, até as mais complexas baseadas em blocos de dados. Dentre todas as estruturas existentes identifica-se um padrão: os dados de um registro estão sempre juntos. Diante do exposto, propõe-se a quebra deste paradigma identificando alguns focos para possíveis ganhos em utilização de espaço de armazenamento e velocidade na obtenção da informação. Para atingir tal meta será desenvolvida uma arquitetura onde cada coluna de cada tabela estará sempre ordenada. Esta organização visa inibir a criação de índices, evitando redundância de dados e tornando toda informação passível de ser localizada com eficiência. Afinal, não se sabe que informação é realmente importante até o momento em que ela se faz necessária.

É necessário testar se a arquitetura proposta pode realmente trazer tais benefícios. Este teste pode ser efetuado através da realização de uma prova de conceito onde são estabelecidas métricas para comparação com outras arquiteturas existentes. Porém, para possibilitar esta comparação, é necessário que esta nova arquitetura esteja implementada. O desenvolvimento de um Sistema Gerenciador de Banco de Dados (SGBD) compreende muitos problemas como controle de acesso, interfaces de comunicação com o usuário, integridade referencial, entre outros. O SGBD MySQL ajuda a resolver este problema, pois tem como característica ser de código aberto, e disponibiliza um mecanismo de construção de extensões, permitindo restringir o desenvolvimento somente à nova organização proposta.

## OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver uma arquitetura para armazenamento e obtenção de dados em um banco de dados relacional.

Os objetivos específicos do trabalho são:

1. construir um tipo de tabela[[1]](#footnote-2) e seu *storage engine* para o MySQL[[2]](#footnote-3);
2. obter estatísticas de desempenho e volume a respeito do tipo de tabela proposto e os tipos InnoDB[[3]](#footnote-4) e MyISAM[[4]](#footnote-5) do MySQL;
3. analisar resultados obtidos e traçar comparativo.

## estrutura do trabalho

O presente trabalho é organizado em cinco capítulos. No primeiro capítulo é apresentada a introdução, os objetivos e estrutura do trabalho. O segundo capítulo fundamenta teoricamente estruturas de dados, sistemas gerenciadores de bancos de dados, tipos de tabela do MySQL, *benchmarks*, a ferramenta *Open Source Database Benchmark* (OSDB) e finaliza com outros trabalhos com temática semelhante. O desenvolvimento do tipo de tabela com seus requisitos, especificação, implementação e resultados são descritos no terceiro capítulo. No quarto, ocorre o desenvolvimento da prova de conceito e é destacada sua especificação, como foi implementada e os resultados obtidos.

# FUNDAMENTAÇÃO TEÓRICA

São contextualizados nas seções seguintes os principais assuntos com o fim de introduzir conceitos utilizados no decorrer do trabalho. Inicialmente são elucidadas algumas prerrogativas em estrutura de dados, SGBDs e suas estruturas especiais. Adiante é explicado o funcionamento dos tipos de tabelas no MySQL, como é utilizado o *benchmarking* como prova de conceito e a ferramenta para sua melhor execução. Por fim, são relacionados trabalhos correlatos ao proposto.

## ESTRUTURAS DE DADOS

Em toda e qualquer aplicação computacional, a relação entre as estruturas de dados empregadas e os algoritmos que as manipulam é sempre muito íntima. Portanto, ao se considerar uma opção para a solução de um problema, estes dois aspectos precisam ser igualmente avaliados, pois os reflexos são mútuos e diretos. (HEINZLE, 2006, p. 3).

Uma estrutura de dados não existe sem um fim específico. Elas são criadas com a intenção de facilitar o entendimento, organizar, economizar recursos ou para melhorar o desempenho de algum algoritmo, ou seja, a existência dessas estruturas de dados está tão assegurada quanto a problemas a serem resolvidos. Alguns problemas nem podem ser resolvidos sem uma estrutura de dados adequada. Heinzle (2006, p. 2) afirma que as estruturas de dados precisam retratar as relações existentes entre os dados de forma a preservar as propriedades do mundo real correspondente, e, ainda, considerar e facilitar as operações ou manipulações que serão posteriormente realizadas sobre eles.

Tipos primitivos, estruturas e bancos de dados se confundem por serem simplesmente uma especialização um do outro. Fisicamente existe apenas um tipo primitivo: o *bit*. Porém, com a evolução das diversas linguagens de programação, cada vez mais estruturas foram sendo abstraídas com a finalidade de auxiliar o programador, surgindo assim tipos primitivos mais complexos como: inteiro, ponto flutuante ou data. A partir desses tipos mais evoluídos são criadas estruturas especializadas que quando agrupadas têm a denominação de banco de dado.

Existem vários tipos de estruturas de dados, entre elas: listas, pilhas, filas, árvores, incluindo algoritmos de pesquisa e ordenação. O registro (*struct* em C) é formato pelo agrupamento de vários tipos primitivos armazenados juntos e linearmente em algum lugar da memória ou do disco.

## SISTEMAS GERENCIADORES DE BANCO DE DADOS

Segundo Elmasri e Navathe (2005, p. 4, grifo do autor), “Um **banco de dados** é uma coleção de dados relacionados”. Um SGBD define-se no próprio nome. São programas que em conjunto formam um sistema que controla e mantém uma coleção de dados, seja esta grande ou pequena, garantindo sua consistência, confiabilidade e segurança.

O modelo relacional tem como principal característica a definição de tabelas e o relacionamento entre elas. Por exemplo, a tabela vendedor tem relacionamento com a tabela vendas, pois, por experiência de negócio do modelador da base de dados, toda venda é feita por um vendedor. Isso denota uma possibilidade de aproximação muito clara entre o modelo e a realidade. Este modelo é o mais utilizado atualmente, tendo como exemplos: PostGreSQL, Firebird, Microsoft SQL Server, entre outros.

### Armazenamento

Ramakrishnan (1998, p. 39) divide a memória de computador hierarquicamente em três níveis: a memória (virtual, principal e *cache* - para acesso rápido a dados); dispositivos mais lentos como discos rígidos; e toda a classe de dispositivos ainda mais lentos como o *Digital Video Disc* (DVD).

Armazenar todos os dados de uma grande empresa no armazenamento primário (memória principal) ainda é muito caro. Porém, existem iniciativas que prevêem que em um futuro próximo será relativamente mais barato manter todos os dados da empresa em memória, pois, além do preço da memória volátil estar diminuindo, o desempenho obtido com esta transição é tão grande que justifica, em alguns casos, o aumento destes custos. Um exemplo dessa transição é o Prevayler, que é um sistema de persistência de dados de código aberto para Java que mantém todos os objetos do persistidos em memória.

A realidade atualmente continua tendo como fardo a utilização em grande escala do armazenamento secundário, deixando ao critério do gerenciador de memória do SGBD descobrir qual a informação mais importante a ser deixada em memória. Portanto, este deve gerenciar adequadamente tanto o armazenamento primário quanto o secundário.

### Arquitetura de armazenamento

Com o fim de otimizar o gerenciamento de armazenamento, o SGBD divide-a em páginas de tamanho fixo, normalmente configurável, de forma semelhante ao que o próprio sistema operacional faz com seu sistema de arquivos. Um dos vários problemas desta arquitetura é identificar de forma rápida quais são as páginas livres e evitar a subutilização do espaço disponibilizado em cada uma destas páginas.

Estas páginas armazenam um ou mais registros. Estes registros podem ter tamanho fixo ou variável. Em um registro de tamanho fixo, todos seus campos (ou atributos), também são de tamanho fixo, ou seja, têm um valor máximo e um mínimo fisicamente delimitado. Já os registros de tamanho variável têm um ou mais campos de tamanho variável, normalmente delimitados por um *byte* terminador (RAMAKRISHNAN, 1998, p. 50). Registros de tamanho variável são otimizados para economizar espaço, porém têm uma perda considerável de desempenho em relação aos de tamanho fixo por ser necessário verificar limites.

O SGBD Oracle e o tipo de tabela InnoDB do MySQL têm estrutura de dados baseada em registros de tamanho variável e cada registro possui um identificador único (*Record IDentification* – RID). Porém, o armazenamento reserva diversos outros problemas como otimização do processo de gravação, reaproveitamento de espaço de registros excluídos, exclusão mútua, segurança e confiabilidade dos dados.

### Índices

A estrutura básica de coleções de dados considerada até este ponto armazena registros de forma aleatória e suporta obtenção de todos os registros ou de um registro específico através de seu RID. Algumas vezes é necessário obter registros informando alguma condição nos campos do registro desejado, por exemplo, achar todos os vendedores com comissão maior que R$ 15,00. Para acelerar a busca desta informação, é possível construir estruturas de dados auxiliares para achar rapidamente os RIDs que satisfaçam a condição informada. Estas estruturas auxiliares são chamadas de índices (RAMAKRISHNAN, 1998, p. 55).

Uma possibilidade de organização dos registros na base de dados seria gravá-los fisicamente de forma ordenada (*clustering*). Se por algum motivo for gravado um registro que se posicione no início de uma tabela, todos os registros subsequentes serão movidos um passo adiante para disponibilizar espaço para este novo registro. Esta estrutura se torna inviável devido ao grande custo de reorganizar repetidamente estes registros fisicamente na base de dados.

Existem várias estruturas de indexação e elas são de responsabilidade de cada SGBD. As mais comuns são a baseada em árvore (Árvore B) e em tabelas de espalhamento (*Hash*). Índices possuem sempre no mínimo dois campos, o campo a ser ordenado e o RID do registro ao qual ele pertence (ELMASRI; NAVATHE, 2005, p. 326).

As tabelas de espalhamento são um tipo de divisão dos registros por categorias. Estas categorias são geradas através do resultado de funções *hash* aplicadas sobre os campos da chave, ou seja, cada busca é efetuada apenas no bloco de dados de uma tabela para onde a categorização apontar. A qualidade desta função *hash* implica diretamente no desempenho da busca. Diferentemente de um índice árvore B, este tipo de índice não é eficiente em consultas por faixa, pois o resultado da função não é necessariamente sequencial.

### Índices Árvore B

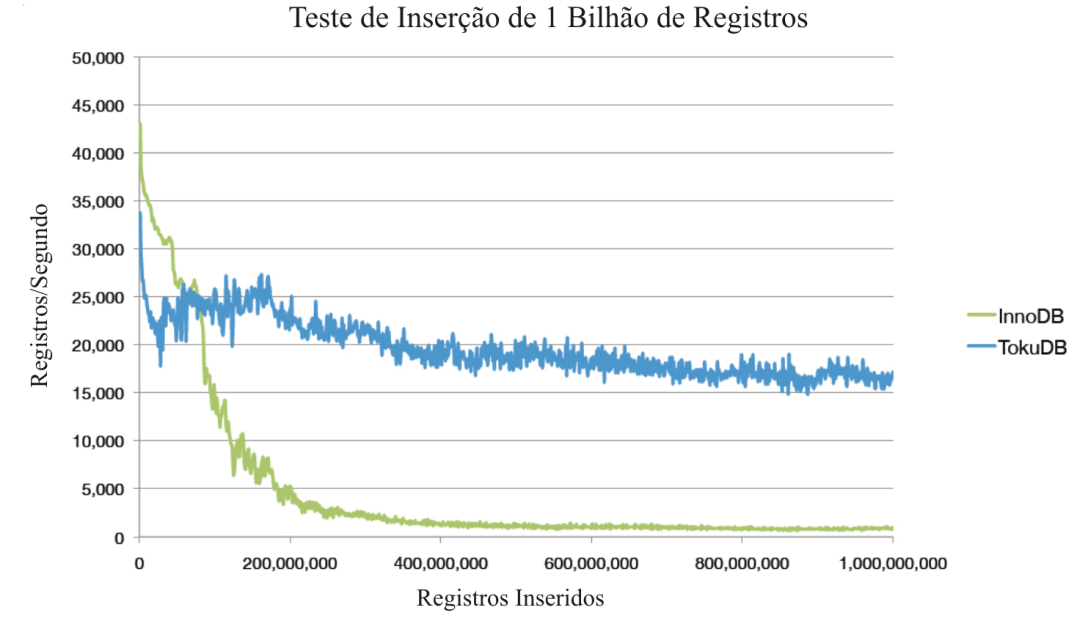
Os índices estruturados em árvore possuem um nó principal chamado raiz, e cada nó filho da raiz pode ter um ou mais nós filhos, e os nós que não possuem filhos, são denominados nós folha. A cada busca, os nós são percorridos e é escolhido entre os nós filhos qual o que mais se assemelha à busca. Esta operação ocorre recursivamente até extinguir as possibilidades de localização ou até o registro ser encontrado.

Garcia-Molina, Ullman e Widom (2001, grifo do autor, p. 166) afirmam que, como o nome implica, uma árvore B organiza seus blocos em uma árvore. A árvore é *equilibrada*, o que significa que todos os caminhos desde a raiz até uma folha têm o mesmo comprimento. Em geral, existem três camadas em uma árvore B: a raiz, uma camada intermediária e as folhas, mas é possível haver qualquer número de camadas.

Uma variante da árvore B, denominada árvore B+ (*B+Tree*), mais comumente utilizada em sistemas comerciais, difere essencialmente na localização dos dados. Nesta última os dados são somente armazenados nos nós folha e nos nós intermediários são apenas chaves que orientam a busca na árvore. Segundo Elmasri e Navathe (2005, p. 336), os nós árvore B são preenchidos entre 50% e 100%, e ponteiros para os blocos de dados são armazenados em ambos os nós internos e nós folhas da estrutura da árvore. Nas árvores B+ os ponteiros para os blocos de dados são armazenados apenas nos nós folhas; Isso pode levar a menos níveis e a índices mais densos.

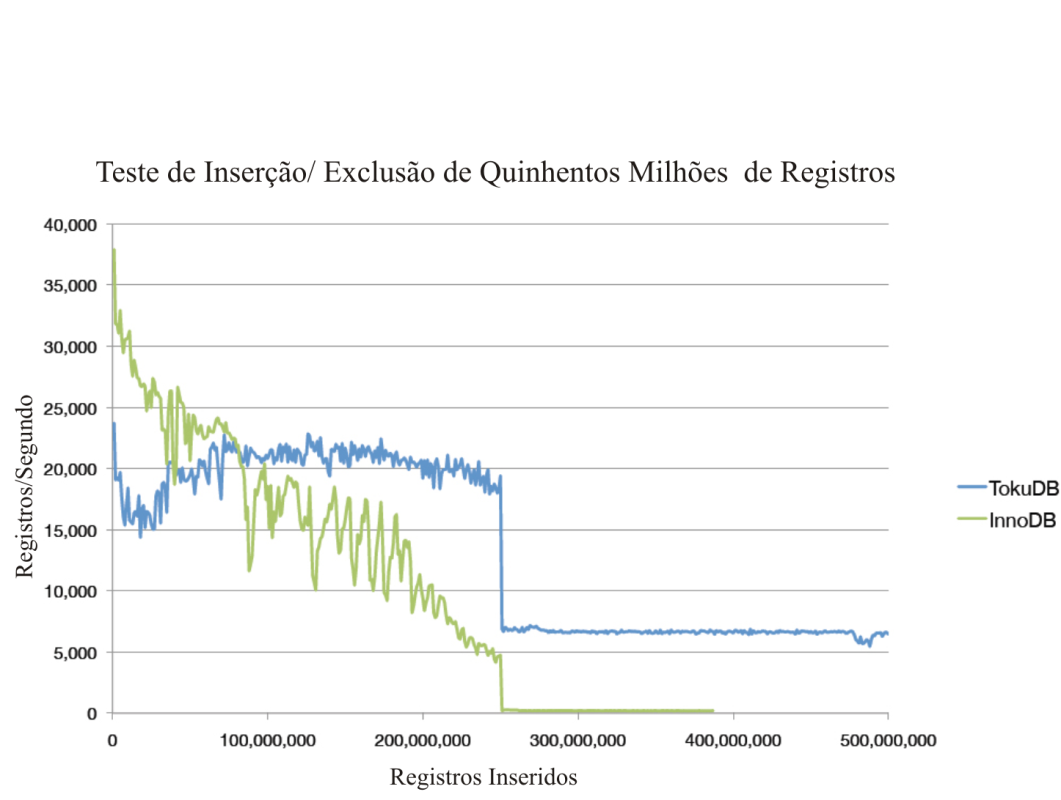
A construção de um índice árvore B leva em consideração o tamanho de cada bloco de dados do banco, sendo este dividido de forma a contabilizar, em relação ao dado a ser organizado, a quantidade de nós filhos que cada nó pode possuir, ou seja, têm uma ramificação com quantidade previsível e fixa de nós filhos. Neste contexto, o espaço das chaves do índice devem sempre prever o maior valor possível de cada campo. Assim, a indexação de um campo texto de tamanho variável sempre ocupará seu tamanho máximo nos nós do índice. Isto pode ser especialmente impactante na indexação de campos de texto de tamanho variável com limites muito grandes.

A empresa Tokutek desenvolveu um tipo de tabela baseado em uma nova tecnologia de indexação em árvores fractais. Este tipo de indexação tem resultados promissores conforme exemplificado na Figura 1 onde é inserido um bilhão de registros em uma tabela com três índices compostos (múltiplas colunas), na Figura 2 onde ocorre a inclusão de duzentos e cinquenta mil registros e a partir disso, a cada mil registros incluídos, são excluídos outros mil, e na Figura 3 onde é feito um teste de consultas com o crescimento na quantidade de registros, e os tempos consulta são extraordinariamente mais constantes. Porém, por mais que seja um tipo de tabela de um banco de dados de código aberto, o TokuDB (tipo de tabela da Tokutek para o MySQL) não tem seu código aberto.



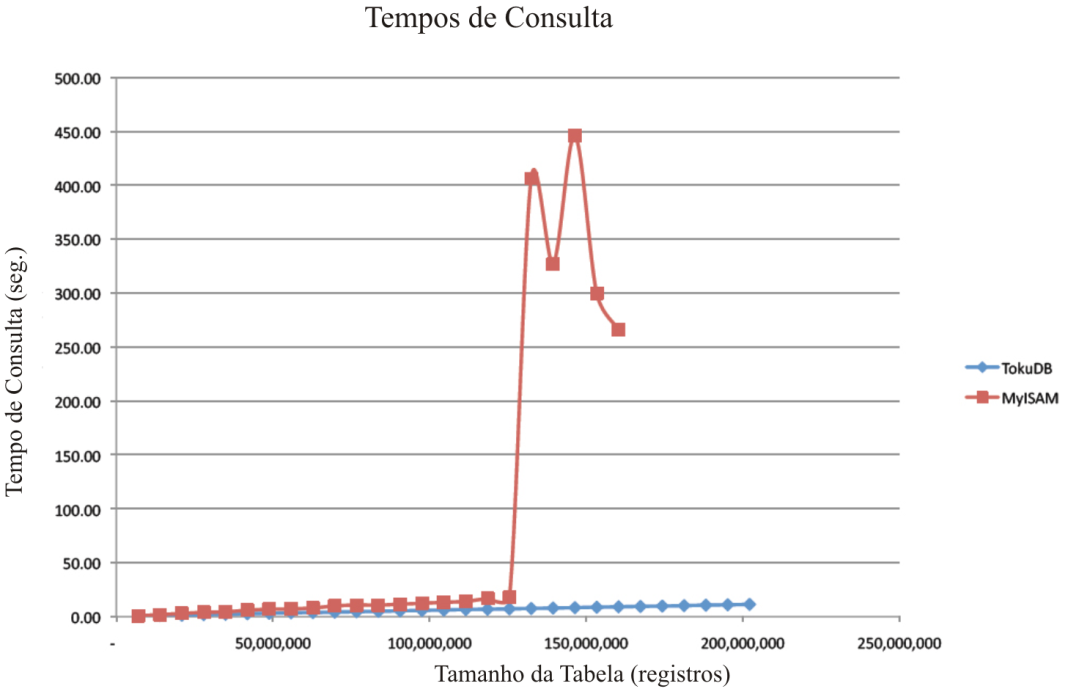
Fonte: adaptado de Tokutek (2009).

Figura - Teste Tokutek: inserção de um bilhão de registros



Fonte: adaptado de Tokutek (2009).

Figura - Teste Tokutek: inserção/exclusão de quinhentos milhões registros



Fonte: adaptado de Tokutek (2009).

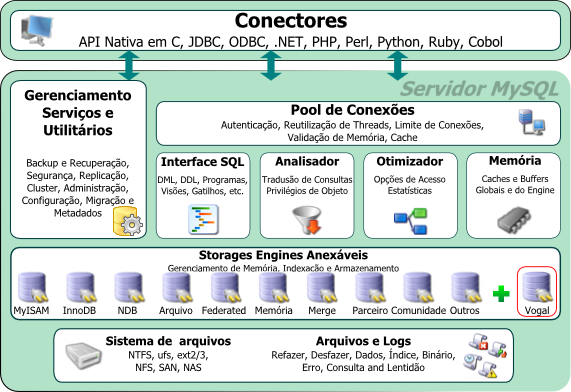
Figura - Teste Tokutek: consulta em tabela com índices de árvores fractais

## tipos de tabela do MySQL

MySQL, o mais popular SGBD com linguagem de consulta estruturada (SQL[[5]](#footnote-6)), é desenvolvido, distribuído e mantido por MySQL AB. MySQL AB[[6]](#footnote-7) é uma empresa comercial, fundada pelos desenvolvedores do MySQL. É da segunda geração de companias *Open Source*, pois une os valores da metodologia de código aberto com um modelo de negócio bem sucedido (DUBOIS, 2008). Empresas como Yahoo!, Google, Nokia e YouTube, utilizam o SGBD MySQL em seus sistemas. É altamente portável, possui alta performance, é confiável e fácil de utilizar (MYSQL AB, 2008).

Os *storage engines* de cada tipo de tabela gerenciam o armazenamento e indexação de dados do MySQL. O servidor comunica-se com o *storage engine* através de uma *Application Programming Interface* (API). Esses podem ser construídos de maneira progressiva começando como somente leitura, e então implementando rotinas de gravação e exclusão, e posteriormente indexação e controle transacional (MYSQL COMMUNITY, 2008).

A partir da versão 5.1, foi introduzida no MySQL uma arquitetura que permite utilizar diferentes *storage engines* adicionando-os em um servidor com o serviço de banco de dados em execução sem precisar recompilar[[7]](#footnote-8) os códigos-fonte do servidor MySQL em si, o que facilita o acoplamento de novos tipos de tabela e evita que o desenvolvedor seja obrigado a possuir o código-fonte completo do MySQL para que seja possível implementar um novo tipo de tabela. É possível visualizar a arquitetura global do MySQL na Figura 4.



Fonte: adaptado de Dubois (2008).

Figura - Arquitetura global do MySQL com *storage engines* plugáveis

### Arquitetura de *storage engines* do MySQL

Alguns dos recursos que podem estar presentes em um *storage engine* são: concorrência, controle transacional, integridade referencial, armazenamento físico, indexação, gerenciamento de memória, dentre outros (DUBOIS, 2008). Estes recursos e qualidade apresentada por cada um são os diferenciais procurados pelo administrador de banco de dados para escolher um tipo de tabela ou outro.

Além dos códigos-fonte dos *storage engines* estarem disponíveis por serem de código aberto, existe uma versão exemplo do MySQL disponibilizada pela MySQL AB na linguagem C++ e um manual de passos a serem seguidos para a sua implementação.

O *storage engine* exemplo não faz nada. Seu propósito é ilustrar ao desenvolvedor como começar a construir um tipo de tabela. O tipo de tabela se chama *EXAMPLE*, disponível junto aos códigos fonte do MySQL, possui a especificação de uma classe com as informações básicas que um *storage engine* deve possuir.

## *BENCHMARKING* COMO Prova de conceito

Um dos métodos de prova de conceito é o *benchmarking*. Este método tem como prerrogativa a execução de testes a partir de sistemas automatizados em dois ou mais programas, fornecendo como saída algumas métricas para comparação entre eles. Segundo Ramakrishnan (1998, p. 465), *benchmarks* deveriam ser portáveis, de fácil compreensão, naturalmente escaláveis para maiores faixas de problemas e deveriam medir performance de pico (por exemplo, transações/segundo) para volumes típicos de trabalho no domínio do sistema.

Dificilmente será criado um comparativo que obtenha resultados simples e diretos. De acordo com as estatísticas e resultados das ferramentas utilizadas para auxiliar o processo de análise, o ser humano é quem chegará às devidas conclusões. São levados em consideração alguns pontos facilmente delineáveis como a capacidade da infra-estrutura, roteiro dos testes, entre outros. Mas também pontos mais subjetivos como otimizações e implementações distintas existentes nos artefatos testados.

Desta forma, o *benchmarking*, como prova de conceito, é utilizado por sua característica sistemática e informacional, possibilitando ao analista projetar, acompanhar e traçar conclusões de menor risco de refutação.

## OSDB

Conforme estudo realizado por Pires, Nascimento e Salgado (2006), o OSDB é uma ferramenta de *benchmark* de código aberto que cresceu a partir de um pequeno projeto de Andy Riebs na Compaq Computer Association com o objetivo inicial de avaliar a taxa de *In/Out* (I/O) e o poder de processamento da plataforma GNU Linux/Alpha[[8]](#footnote-9). O OSDB é compatível com vários SGBDs, entre eles o MySQL, e possibilita a comparação através de métricas como tamanho máximo suficiente do banco de dados para completar o teste em menos de 12 horas, tempo de resposta das consultas e número de linhas retornadas por segundo.

*Benchmarks* são raramente confiáveis e úteis. Estes são vistos com desconfiança, pois os indivíduos que apresentam os resultados podem ter motivações e competência suspeitas. Também são considerados inúteis, pois dificilmente o modelo é equiparável aos trabalhos que se deseja comparar (RIEBS, 2004). Portanto, para um *benchmark* estar acima de qualquer suspeita e sempre se adequar a necessidade do desenvolvedor, o ideal é ser transparente e acessível a qualquer interessado, como o OSDB que é de código aberto.

## trabalhos correlatos

Existem diversas iniciativas correlatas ao trabalho proposto, sendo: um novo tipo de tabela com o intuito de substituir o MyISAM, o Maria do MySQL (DUBOIS, 2008); o tipo de indexação *Bitmap*, que é um método aplicável apenas para algumas necessidades específicas (CYRAN et al., 2005); e um comparativo de desempenho entre SGBDs de código aberto (PIRES; NASCIMENTO; SALGADO, 2006).

### Maria: tipo de tabela

O *storage engine* do tipo de tabela Maria foi introduzido no MySQL a partir da versão 6.0.6 e é uma especialização do MyISAM com o recurso de operações atômicas (operações iniciam bloqueando a tabela para alteração e só liberam outras alterações com o término da operação corrente, com o desbloqueio da tabela). Este tipo de tabela suporta todas as principais funcionalidades do *storage engine* MyISAM, porém inclui facilidades de recuperação (em caso de um problema de sistema), gravação completa de históricos (incluindo criação e limpeza de tabelas), entre outros. O Maria mantém a velocidade e a flexibilidade de seu predecessor combinando estas características com suporte transacional (DUBOIS, 2008).

No MyISAM existem três formatos de linha: fixo, variável e comprimido. O primeiro e o segundo são selecionados automaticamente dependendo do tipo dos campos da tabela, ou seja, caso a tabela tenha algum campo de tamanho variável o formato da linha muda para atender essa necessidade. O formato de linha comprimida, que é uma configuração opcional que visa redução de utilização de espaço em disco, possui um processo intermediário de compressão e descompressão. Porém estes três formatos de linha existem por compatibilidade e para serem usados em modelos não transacionais. Portanto, foi implementado um novo formato de linha no tipo de tabela Maria que é formado por páginas, exatamente para possibilitar um fim transacional otimizado.

### Indexação *Bitmap*

O propósito de um índice é prover ponteiros para as linhas de uma tabela correspondentes a uma chave. Em um índice regular, isso é alcançado gravando uma lista de RIDs para as chaves de cada linha. O SGBD Oracle grava as chaves repetidamente para cada RID. Em um índice *Bitmap*, é utilizado um mapeamento de *bits*, ao invés de uma lista de RIDs (CYRAN et al., 2005). Cada *bit* deste mapeamento corresponde a um possível RID, ou seja, se o *bit* estiver ligado, a posição dele corresponde à posição do registro desejado na tabela (Figura 5). Isso é eficiente pelo fato de não precisar acessar os dados físicos da tabela para restringir as informações que serão acessadas. O sistema de indexação permite a realização de intersecção entre índices do tipo *Bitmap* sobre diferentes colunas para aumentar ainda mais a performance e a pontualidade da obtenção da informação.

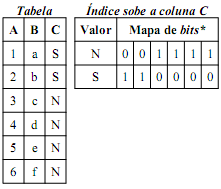


Figura - Tabela com índice do tipo *Bitmap*

Porém, esta eficiência tem um alto custo de atualização. Para cada gravação efetuada na tabela, existe a necessidade de reconstrução total do índice. Isso se faz imprescindível pela necessidade de atualização do mapa de *bits*, tornando-se inviável para tabelas com grande volume de atualização. Portanto, este tipo de índice deve ser utilizado sobre colunas com muita repetição do conteúdo e sobre tabelas com baixo índice de atualização.

### Comparativo de desempenho entre bancos de dados de código aberto

O objetivo da proposta de Pires, Nascimento e Salgado (2006, p. 1) é apresentar um estudo comparativo de desempenho entre os SGBD MySQL e PostGreSQL, em plataforma GNU/Linux, utilizando o *Open Source Database Benchmark* (OSDB). O estudo consiste em analisar as métricas geradas pelo OSDB e estimular melhorias nas funcionalidades dos SGBDs relacionadas com desempenho.

É inicialmente feita uma análise entre alguns *benchmarks*, chegando-se à conclusão de que o OSDB é o que melhor se adéqua às suas necessidades de testes pela abrangência destes testes e possibilidades de variação.

As configurações de hardware, sistema operacional e banco de dados em que seriam realizados todos os testes são descritas de forma detalhada. Estes testes são resumidos visando auxiliar a análise e a inferência de um resultado. Em seguida são apresentados detalhadamente apenas os testes onde houve maior divergência. Alguns destes testes são:

1. carga e estrutura: criação de tabelas, carga das tabelas e criação de índices;
2. seleções: capacidade do SGBD de escolher a melhor forma de consulta à tabela;
3. junções: teste de junção sem utilizar índices;
4. projeções: seleção utilizando uma cláusula de unicidade de linhas (cláusula distinct do SQL);
5. agregações: utilização de agrupamento (cláusula group by) e funções de agregação em colunas indexadas (por exemplo, a função sum do SQL).

Como conclusão é definida onde há possibilidade de melhoria em cada um dos SGBDs e qual que apresentou melhores resultados nos testes executados. São então descritos possíveis novos testes em outras configurações de hardware, sistema operacional e SGBD.

# DESENVOLVIMENTO DO TIPO DE TABELA

O desenvolvimento do projeto divide-se em duas etapas: o desenvolvimento do Tipo de Tabela e o desenvolvimento da prova de conceito. Neste capítulo são apresentados os requisitos do sistema, especificação, implementação e os resultados obtidos em relação ao desenvolvimento do Tipo de Tabela.

## requisitos principais do problema a ser trabalhado

Simplificando a estrutura de uma tabela a uma estrutura formada por linhas e colunas onde cada linha é um registro e cada coluna é um campo, é representada na Figura 6 esta estrutura com uma exemplificação de como ficariam os índices, caso estes existissem. Nesta estrutura o registro é atômico (linha em destaque na ), ou seja, está sempre junto fisicamente. Na Figura 7 é representado superficialmente como será a nova estrutura de tabelas, tornando cada campo independente, ordenado e inter-relacionável.

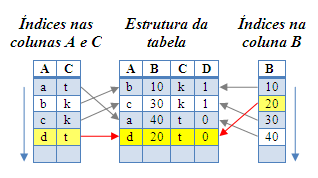


Figura - Estrutura comum de tabela

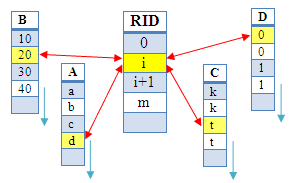


Figura - Nova estrutura de tabela

A arquitetura do Tipo de Tabela proposto deverá:

1. permitir ao usuário[[9]](#footnote-10) criar e remover tabelas (Requisito Funcional - RF);
2. permitir ao usuário inserir, atualizar e remover registros, das tabelas criadas (RF);
3. permitir ao usuário extrair registros, das tabelas criadas, utilizando a linguagem SQL, limitando-se a uma tabela por extração e sem recursos de agrupamento (RF);
4. permitir ao usuário projetar quaisquer campos da tabela desejada e criar condições para restringir a extração (RF);
5. ser implementado utilizando a linguagem de programação C++ com a biblioteca GNU *Compiler Collection* (GCC) em Linux com o pacote Ubuntu (Requisito Não Funcional – RNF);
6. ser implementado compreendendo o armazenamento dos dados e metadados[[10]](#footnote-11) no dispositivo secundário e implementando um gerenciamento de memória primitivo, não implementando controle transacional nem de concorrência (RNF);
7. As instruções SQL devem obedecer ao padrão definido no manual de referência do MySQL (DUBOIS, 2008) (Regra de Negócio - RN);
8. As tabelas devem ser definidas apenas com campos de tipo texto ou numérico inteiro (RN);
9. Os tipos texto serão limitados a 127 caracteres (RN);
10. Os tipos numéricos deverão estar entre -2147483648 e 2147483647 (inteiro de 32 *bits*) (RN);
11. As limitações de tamanho na definição do campo não serão validadas (RN);
12. As tabelas devem ser definidas apenas com os campos e seus tipos. Qualquer outra definição ocasionará erro de criação. Como por exemplo, chaves primárias, obrigatoriedade, auto-incremento, etc. (RN).

## ESPECIFICAÇÃO

O projeto do Tipo Tabela é especificado empregando o processo de análise e projeto orientado a objetos. A ferramenta Enterprise Architect (EA) é utilizada para o desenvolvimento de todos os diagramas da *Unified Modeling Language* (UML).

O Tipo de Tabela foi denominado Vogal (AEIOU - Armazenamento e Extração de Informação Otimizada por Unicidade) e seu código fonte está disponível no endereço <http://code.google.com/p/vogal/>.

### Diagrama de casos de uso

Na são demonstrados os casos de uso do projeto.



Figura - Diagrama de casos de uso

Os casos de uso visualizados na são:

1. Criar tabela: permite ao usuário criar uma tabela (Quadro 1);
2. Remover tabela: permite ao usuário remover uma tabela (Quadro 2);
3. Inserir registro: permite ao usuário inserir um registro (Quadro 3);
4. Atualizar registro: permite ao usuário atualizar um registro (Quadro 4);
5. Remover registro: permite ao usuário remover um registro (Quadro 5);
6. Selecionar registro: permite ao usuário selecionar um registro (Quadro 6).

|  |  |
| --- | --- |
| Caso de Uso | Criar tabela |
| Resumo | Através de um cliente[[11]](#footnote-12) o usuário redige uma instrução SQL para criação de uma tabela do tipo Vogal. |
| Ator | Usuário |
| Pré-condição | O usuário deve estar conectado a um banco de dados MySQL existente através de qualquer cliente que permita instruções SQL. |
| Pós-condição | Tabela criada no banco de dados. |
| Fluxo principal | 1. O usuário cria uma instrução *Data Definition Language* (DDL) [[12]](#footnote-13) para criação de uma tabela do tipo Vogal; 2. O usuário comanda a execução desta instrução DDL no servidor através de opção disponível no cliente; 3. O sistema cria a tabela no esquema. |
| Exceções | 1. O sistema gera erro na criação de uma tabela caso outra tabela possua mesmo nome; 2. O sistema gera erro na criação de uma tabela caso a tabela tenha colunas com nomes iguais. |

Quadro - Caso de uso Criar tabela

|  |  |
| --- | --- |
| Caso de Uso | Remover tabela |
| Resumo | Através de um cliente o usuário redige uma instrução SQL para remoção de uma tabela do tipo Vogal. |
| Ator | Usuário |
| Pré-condição | O usuário deve estar conectado a um banco de dados MySQL existente através de qualquer cliente que permita instruções SQL. |
| Pós-condição | Tabela removida do banco de dados. |
| Fluxo principal | 1. O usuário cria uma instrução DDL para remoção de uma tabela do tipo Vogal; 2. O usuário comanda a execução desta instrução DDL no servidor através de opção disponível no cliente; 3. O sistema remove a tabela no esquema. |
| Exceções | 1. O sistema gera erro na remoção de uma tabela que não existe. |

Quadro - Caso de uso Remover tabela

|  |  |
| --- | --- |
| Caso de Uso | Inserir registro |
| Resumo | Através de um cliente o usuário redige uma instrução SQL para inserção de um registro em uma tabela do tipo Vogal. |
| Ator | Usuário |
| Pré-condição | O usuário deve estar conectado a um banco de dados MySQL existente através de qualquer cliente que permita instruções SQL. |
| Pós-condição | Registro inserido na tabela. |
| Fluxo principal | 1. O usuário cria uma instrução *Data Manipulation Language* (DML) [[13]](#footnote-14) para inserção de um registro em uma tabela do tipo Vogal; 2. O usuário comanda a execução desta instrução DML no servidor através de opção disponível no cliente; 3. O sistema insere o registro na tabela. |
| Exceções | 1. O sistema gera erro na inclusão do registro caso a tabela não exista. |

Quadro - Caso de uso Inserir registro

|  |  |
| --- | --- |
| Caso de Uso | Atualizar registro |
| Resumo | Através de um cliente o usuário redige uma instrução SQL para atualização de um registro em uma tabela do tipo Vogal. |
| Ator | Usuário |
| Pré-condição | O usuário deve estar conectado a um banco de dados MySQL existente através de qualquer cliente que permita instruções SQL. |
| Pós-condição | Registro atualizado na tabela. |
| Fluxo principal | 1. O usuário cria uma instrução DML para atualização de um registro em uma tabela do tipo Vogal; 2. O usuário comanda a execução desta instrução DML no servidor através de opção disponível no cliente; 3. O sistema exclui o registro na tabela; 4. O sistema insere o registro na tabela. |
| Fluxo alternativo | A partir do item 2 do fluxo principal:   * 1. O sistema não consegue localizar o registro de acordo com a instrução recebida;   2. O sistema não exclui nem insere o registro na tabela. |
| Exceções | 1. O sistema gera erro na atualização do registro caso a tabela não exista. |

Quadro - Caso de uso Atualizar registro

|  |  |
| --- | --- |
| Caso de Uso | Remover registro |
| Resumo | Através de um cliente o usuário redige uma instrução SQL para remoção de um registro em uma tabela do tipo Vogal. |
| Ator | Usuário |
| Pré-condição | O usuário deve estar conectado a um banco de dados MySQL existente através de qualquer cliente que permita instruções SQL. |
| Pós-condição | Registro removido na tabela. |
| Fluxo principal | 1. O usuário cria uma instrução DML para remoção de um registro em uma tabela do tipo Vogal; 2. O usuário comanda a execução desta instrução DML no servidor através de opção disponível no cliente; 3. O sistema localiza o registro de acordo com a instrução recebida; 4. O sistema remove o registro da tabela. |
| Fluxo alternativo | A partir do item 2 do fluxo principal:   * 1. O sistema não consegue localizar o registro de acordo com a instrução recebida;   2. O sistema não remove o registro na tabela. |
| Exceções | 1. O sistema gera erro na remoção do registro caso a tabela não exista. |

Quadro - Caso de uso Remover registro

|  |  |
| --- | --- |
| Caso de Uso | Selecionar registro |
| Resumo | Através de um cliente o usuário redige uma instrução SQL para seleção de um registro em uma tabela do tipo Vogal. |
| Ator | Usuário |
| Pré-condição | O usuário deve estar conectado a um banco de dados MySQL existente através de qualquer cliente que permita instruções SQL. |
| Pós-condição | Registro removido na tabela. |
| Fluxo principal | 1. O usuário cria uma instrução DML para remoção de um registro em uma tabela do tipo Vogal; 2. O usuário comanda a execução desta instrução DML no servidor através de opção disponível no cliente; 3. O sistema localiza o registro de acordo com a instrução recebida; 4. O sistema remove o registro da tabela. |
| Fluxo alternativo | A partir do item 2 do fluxo principal:   * 1. O sistema não consegue localizar o registro de acordo com a instrução recebida;   2. O sistema não remove o registro na tabela. |
| Exceções | 1. O sistema gera erro na remoção do registro caso a tabela não exista. |

Quadro - Caso de uso Selecionar registro

### Diagrama de classes

A Figura 9 ilustra o diagrama de classes do *storage engine* Vogal e para valorizar o aspecto legibilidade foram omitidos os atributos de cada classe e seus métodos. Neste diagrama há uma classe destacada, a classe handler. Esta classe é a interface entre o MySQL com o *storage engine*, ou seja, nela estão representados os métodos que são suportados e necessários para que integração ocorra corretamente e na classe ha\_vogal estes métodos são implementados.

Em seguida são detalhadas algumas classes consideradas essenciais para o sistema. No apêndice A o diagrama representado na Figura 9 pode ser consultado sem omissões.



Figura - Diagrama de classes do *storage engine*

#### Classes de gerenciamento

As classes de gerenciamento são: ha\_vogal (Figura 10), vogal\_definition (Figura 12), vogal\_manipulation (Figura 11), vogal\_storage (Figura 13) e vogal\_cache (Figura 14).

A classe ha\_vogal é a implementação da interface de comunicação com a API dos *storage engines*. Toda requisição do banco de dados passa por este componente que gerencia a comunicação entre o banco de dados com as classes especializadas: vogal\_definition (definição das tabelas e organização destas informações dentro dos blocos de dados), vogal\_manipulation (manipulação dos registros nas tabelas), vogal\_storage (armazenamento e recuperação dos blocos em disco) e vogal\_cache (otimização de memória). Ao receber a chamada open da API do *storage engine* a classe ha\_vogal aciona o método openDatabase. Este, caso o arquivo do banco de dados não exista, cria-o através da classe vogal\_storage e então inicia o processo de criação do dicionário de dados através do método createDataDictionary que gerencia a interação entre as classes vogal\_storage e vogal\_definition. Assim que o dicionário de dados estiver criado, ou o arquivo de banco de dados já existir, o método openDatabase abre este arquivo e inicia o processo initialize que inicializa a operação de gerenciamento de memória na classe vogal\_cache que por sua vez inicializa o método bufferize que carrega em memória informações relevantes ao banco de dados para otimizar seu desempenho.



Figura - Definição da classe ha\_vogal e interface handler



Figura - Definição da classe vogal\_manipulation



Figura - Definição da classe vogal\_definition



Figura - Definição da classe vogal\_storage



Figura - Definição da classe vogal\_cache

Os métodos da interface que representam alguma interação com índices não serão implementados, pois o projeto Vogal não permite a criação de índices secundários e o mecanismo de indexação das colunas implementado pelo sistema é apresentado à API como um registro atômico comum, ou seja, para a API o Tipo de Tabela Vogal é uma coleção de registros sem índices.

A partir do momento em que é executado o método openDatabase da classe ha\_vogal, fica a critério do usuário que operações ele deseja realizar no banco, ou seja, as inserções, exclusões, criação de tabelas, etc. ocorrem na ordem que o usuário desejar, impossibilitando a definição de uma sequência lógica.

Quando usuário criar uma instrução DDL de criação de tabela em um cliente, o método create da API é chamado e então o método newTable da classe vogal\_definition é disparado. Este método verifica no dicionário de dados, utilizando informações da classe vogal\_cache, se a tabela existe ou não e caso não exista, cria a tabela através do método createTableStructure, o qual obtém o espaço de um bloco de memória e define a estrutura da tabela e repassa essa informação para a classe vogal\_storage para persistência. Da mesma forma, caso o método delete\_table da API for executado, é então chamado o método dropTable da classe de vogal\_definition, a qual localiza a tabela no banco e viabiliza sua remoção.

Se o usuário executar uma instrução DML para qualquer operação sobre algum registro, é executado o método open da API. Este por sua vez, mediado pela classe ha\_vogal, chama o método openTable da classe vogal\_definition, o qual abre as tabelas do dicionário de dados para verificar a existência da tabela e obter sua definição e ponteiros para localização da tabela e suas colunas dentro do arquivo do banco de dados. Existindo a tabela, através da vogal\_manipulation é executado a escrita e leitura do registro de acordo com a operação solicitada. No caso da exclusão e consulta a leitura dos índices é feita para localização do registro, e se a operação for de exclusão, este é removido. No caso da inserção e atualização é feito uma leitura nos índices pra escrita no local apropriado de cada coluna dentro de seu índice, se for atualização, é feito primeiro a operação de exclusão do registro. Estas operações ocorrem através da classe vogal\_manipulation e principalmente pelos métodos openCursor, fetch, insertData, dropRid, readRead, writeRid, entre outros.

### Estrutura do arquivo de dados

O tipo de tabela Vogal é basicamente um grande arquivo de 100MB dividido em cem mil blocos de 1KB (Figura 15). Cada bloco possui um cabeçalho e um corpo (Figura 16). No cabeçalho é definido o tipo do bloco e se ele é válido ou não. Quando um arquivo de dados for inicializado todos os blocos são marcados com não válidos, e a partir do momento em que vão sendo utilizados, são marcados como válidos. A partir do momento que um bloco não é mais utilizado ele é marcado novamente como não válido. Essa informação de validade do bloco é importante para, ao inicializar o banco de dados, sejam identificados quais blocos estão livres.

Existem dois tipos de bloco: tabela e coluna. No corpo de ambos os tipos bloco existe uma árvore B. Em decorrência da variabilidade do tamanho dos registros há a necessidade de manter a informação de quantidade que significa a quantidade de nós da árvore que estão no nó bloco atual.

Quando o tipo de bloco for tabela a árvore é ordenada pelo RID (Figura 17) que mantém junto ao nó do RID as informações de deslocamentos de suas colunas. Quanto o tipo de bloco for coluna a árvore é ordenada pelo valor da coluna e que é complementada pelo RID do registro (Figura 18). Possibilitando assim o inter-relacionamento.

A informação de deslocamento das colunas é formada pelo deslocamento físico do bloco no arquivo de dados complementado pela informação de deslocamento do nó da árvore no bloco.

|  |  |  |
| --- | --- | --- |
| Bloco 1 | Bloco 2 | Bloco 3 |
| Bloco 4 | Bloco 4 | Bloco 5 |
| Bloco 6 | ... |  |

Figura - Estrutura do arquivo de dados

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Cabeçalho**   |  |  | | --- | --- | | Válido? | Tipo do Bloco | | **Corpo**   |  |  | | --- | --- | | Qtd. | Árvore B de registros ou RIDs | |  |

Figura - Estrutura de um bloco genérico

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Esquerda | | RID 1 | | Deslocamentos | | | | Esquerda | |
| RID 2 | Deslocamentos | | | | Esquerda | | RID 2 | | Deslo- |
| -camentos | | | Direita | | |  | | | |
|  | |  | | | | | | | |

Figura - Estrutura da árvore de registros

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Esquerda | | Chave 1 | | | | RID | | Esquerda |
| Chave 2 | | | RID | | Esquerda | | Chave 3 | |
| RID | Direita | | |  | | | | |
|  | |  | | | | | | |

Figura - Estrutura da árvore de RIDs

Todos os dados gravados na base de dados são de tamanho variável, ou seja, a gravação do valor 1 em um campo numérico com precisão 8 ocupará apenas 2 *bytes*. Isso ocorre porque todos os dados têm 1 *byte* de identificação de tamanho. Este *byte* tem 7 *bits* de informação de tamanho e 1 *bit* que identifica se deve ser lido um novo byte de definição de tamanho. Dessa forma, com 1 *byte* podemos medir campos de até 127 *bytes* (27), e com 2 *bytes*, podemos medir campos de até 16384 *bytes* (214), e assim por diante, com um tamanho teoricamente ilimitado.

Como o objetivo da proposta é evitar redundância e desperdício de espaço em disco, o espaço que seria subutilizado é, dessa forma, diminuído drasticamente. Como a operação de conversão destes dados ocorre em memória o ganho de desempenho na leitura da base de dados potencialmente supera a perda na verificação dos limites. A indexação em árvore B foi preferida em relação a árvore B+ em virtude da necessidade de menor redundância e complexidade.

O dicionário de dados segue ao mesmo padrão de estrutura e é formado pelas tabelas:

1. OBJS: coleção dos objetos do Tipo de tabela, neste caso tabelas, composta pelas colunas NAME (Nome do objeto), TYPE (Tipo do objeto. Exemplo: TABLE) e LOCATION (Localização do bloco inicial da árvore de dados da tabela no arquivo de dados);
2. COLS: coleção das colunas das tabelas do Tipo de Tabela, composta pelas colunas TABLE\_RID (RID da tabela mãe da coluna), NAME (Nome da coluna), TYPE (Tipo da coluna. Exemplo: NUMBER) e LOCATION (Localização do bloco inicial da árvore de dados da coluna no arquivo de dados).

## IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas, parte do código desenvolvido e a operacionalidade da implementação.

### Técnicas e ferramentas utilizadas

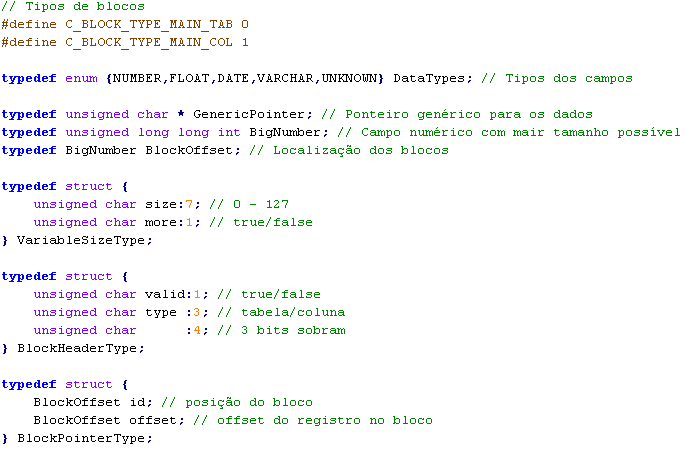
Todo o ambiente de desenvolvimento se encontra em uma máquina virtual disponível através da ferramenta VirtualBox da Sun Microsystems. Esta máquina virtual tem o Linux instalado na distribuição do Ubuntu. O protótipo, independente do MySQL, foi implementado em C no ambiente Eclipse. Deste protótipo surgiram as principais estruturas que foram integradas ao MySQL utilizando o editor de texto VIsual (VI) que está disponível em praticamente todas as distribuições do sistema operacional Linux e o editor visual Geany. O código fonte foi compilado utilizando o Make e o GCC, todos disponibilizados junto à distribuição do Linux. A depuração, quando necessária, foi efetuada através da ferramenta Data Display Debugger (DDD) que implementa um ambiente visual de depuração através do utilitário GNU DeBbuger (GDB). O aplicativo Bless auxiliou no processo de análise do arquivo de dados em formato hexadecimal. As figuras dos códigos fonte apresentados neste trabalho foram feitas usando o editor Notepad++.

O MySQL na sua versão 5.1.34-0 foi primordial para implementação pela infra-estrutura por causa da existência de *storage engines* e pelos testes por causa da possibilidade acesso ao banco de dados. Como ferramenta de execução de instruções SQL no banco de dados foi utilizada o MySQL Navigator.

### Código desenvolvido

É importante salientar que os códigos apresentados nos quadros deste tópico podem ter sofrido modificações no seu leiaute quando comparados ao código original visando destacar alguns aspectos, porém sem nunca perderem sua lógica básica de funcionamento.

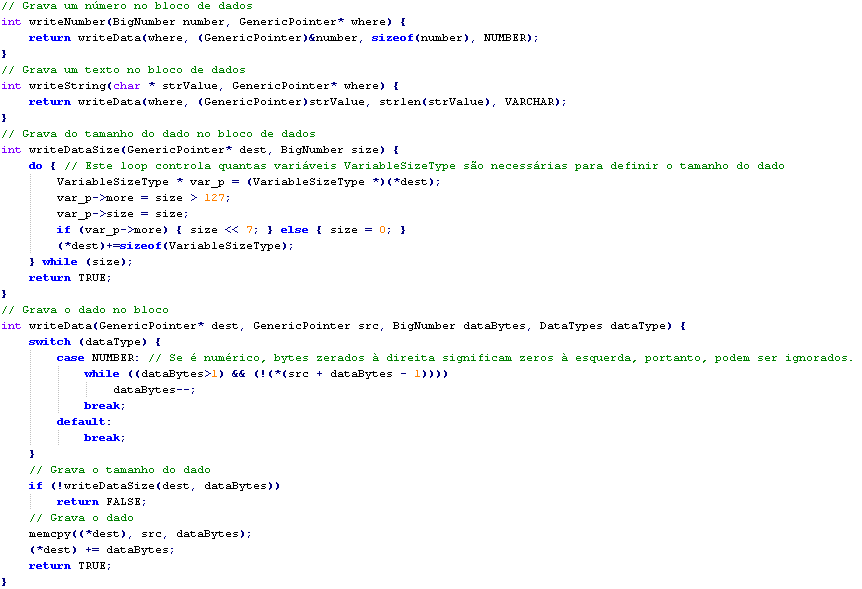
O desenvolvimento iniciou com a implementação de um protótipo que abrangia grande parte dos requisitos do sistema. Porém, não integrado à API dos *storage engines* do MySQL. Neste protótipo foram criadas as estruturas de controle e tipos padrão para utilização no sistema (Quadro 7). Estas estruturas são importantes, pois não foram modeladas como classes e são utilizadas como tipos básicos em grande parte do sistema.



Quadro - Estruturas de controle e tipos padrão

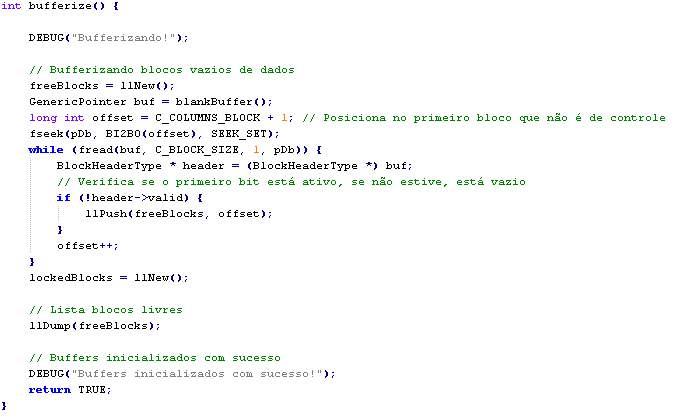
Antes de implementar as classes gerenciadoras do sistema, há a necessidade de criação de mecanismo de coleções. Foram criados quatro tipos de coleção: lista de faixas (LinkedListRoot), lista de valores (ValueListRoot), lista de nomes e valores (PairListRoot) e árvores (StringTreeRoot). A lista de faixas procura otimizar listas de números, economizando memória e melhorando o desempenho de busca e atualização. A lista sequencial é uma lista dinâmica simples cuja funcionalidade foi estendida na lista de nomes e valores. A árvore é uma estrutura de árvore binária em memória que tem um nome como chave.

A primeira necessidade em uma estrutura de armazenamento é a definição de seu dicionário de dados e existe um problema relativo à sua criação. Este problema de criação ocorre pois o dicionário de dados é a informação básica para criação de todas as estruturas, ou seja, não é possível criar uma rotina de criação de tabelas e utilizá-la para criar o dicionário de dados pois ela, por definição, consulta o dicionário de dados para criar novas estruturas. Desta forma é necessário que o dicionário de dados seja criado de forma independente do método de criação de tabelas. Mas antes de criar o dicionário de dados, há a necessidade de definição do formato de escrita e leitura destas informações. Os principais métodos de escrita são exemplificados no Quadro 8. Os métodos de leitura são análogos aos de escrita e são o readNumber, readString, readSizedDada e readData e estão na classe vogal\_storage.



Quadro - Métodos de escrita de dados nos blocos

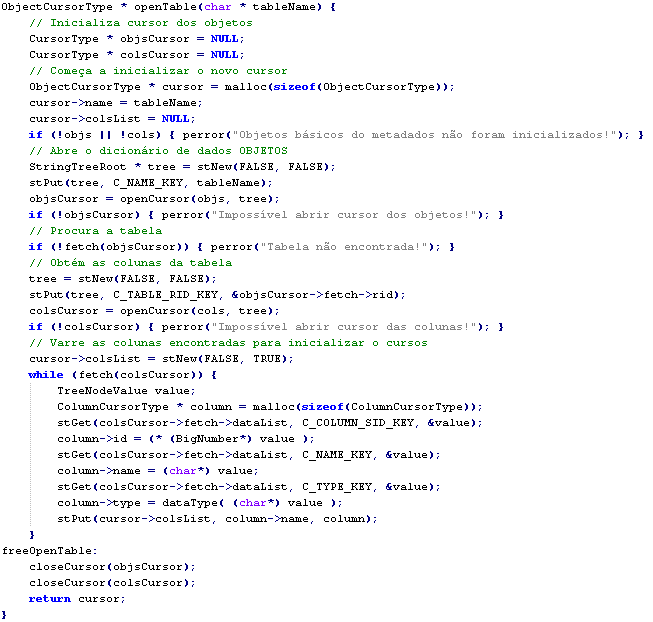
Definido os métodos de escrita e leitura pode-se criar o dicionário de dados. Em seguida deve ser coletadas informações do arquivo de banco de dados criado de forma a otimizar o desempenho. No Quadro 9 é exemplificado o método onde são lidos todos os blocos do arquivo e armazena essa informação em uma lista de faixa.



Quadro - Método de coleta dos blocos livres

Em diversas partes do sistema é necessário reservar alguns blocos para gravação e evitar que outros métodos utilizem-no antes que o método solicitante efetive sua gravação. Esta reserva é feita na classe vogal\_storage nos métodos lockFreeBlock (reserva bloco) e unlockBlock (cancela reserva do bloco). A reserva é feita movimentado o bloco solicitado entre duas coleções na classe vogal\_cache: freeBlocks (lista de faixas de blocos livres) e lockedBlocks (lista de faixas de blocos reservados).

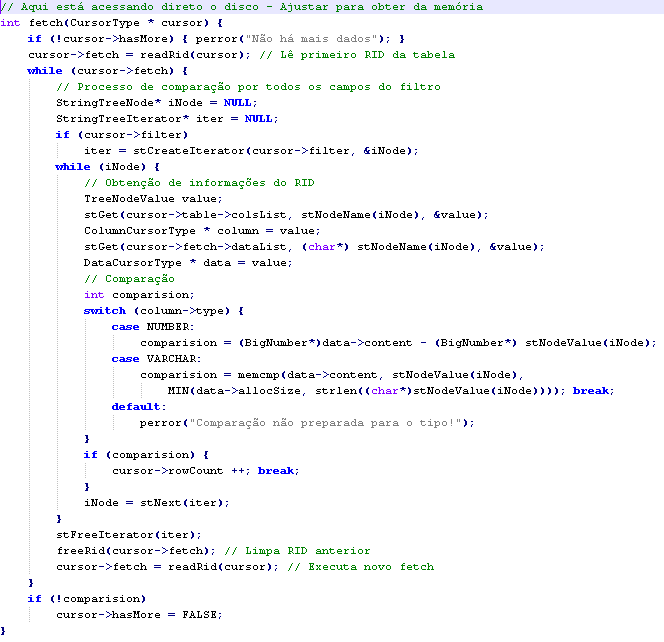
Uma vez o dicionário de dados criado e o método de escrita definido é possível efetuar todas as operações ao qual este trabalho se propõe: criar e remover tabelas, consultar, inserir, alterar e remover registros. Antes de qualquer uma destas operações o método de openTable da classe vogal\_definition (Quadro 10). No caso da criação de tabelas, este método é chamado para garantir sua inexistência. Na remoção de tabelas, exatamente o contrário. Porém, nas outras operações este método é essencial para organizar em memória as definições da tabela na classe ObjectCursorType, as informações que foram previamente persistidas no armazenamento secundário. Desta forma, a tabela de tabelas, denominada OBJS, e a tabela de colunas, denominada COLS, são acessadas como tabelas normais e podem ser consultadas inclusive pelo usuário final.



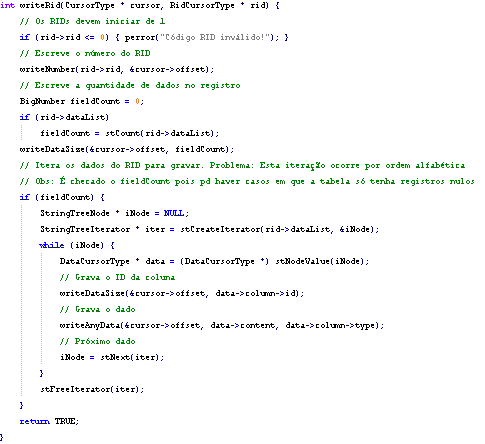
Quadro - Definição do método de abertura de tabelas

Com a tabela aberta é possível efetuar as operações desejadas sobre ela. Para consultas, alterações e exclusões de registros é chamado o método fetch (Quadro 11) da classe vogal\_manipulation. Este método movimenta o objeto ObjectCursorType pela tabela até localizar o registro desejado sendo então possível efetivar a operação desejada. Caso a operação seja apenas consulta, o método rnd\_next da API será chamado para efetuar a consulta de cada registro obtido e o ha\_vogal gerenciará adequadamente a chamada.

Os métodos de escrita e leitura são principalmente o writeRid (Quadro 12) e o readRid (Quadro 13), respectivamente, e estão localizados na classe vogal\_manipulation.



Quadro - Definição do método para consulta de registros



Quadro - Definição do método para escrita de registros



Quadro - Definição do método de leitura do registro

Existem outros métodos de suma importância utilizados no sistema Vogal, porém, de menor relevância à tecnologia proposta, portanto, aqui, não destacados.

### Operacionalidade da implementação

Inicialmente o usuário deve acessar um cliente que disponibilize uma interface de comunicação com um servidor MySQL e permita a elaboração de instruções SQL, e conectar a um banco de dados previamente criado. O servidor MySQL ao qual o cliente estiver conectado deve dar suporte ao Tipo de Tabela Vogal.

Para criar uma tabela o usuário deve elaborar uma instrução DDL semelhante ao definido no Quadro 14 e comandar a execução da instrução no servidor. É de suma importância que o usuário informe o tipo de tabela da tabela a ser criada (Cláusula TYPE = Vogal), caso contrário a tabela será criada com o tipo de tabela padrão do MySQL (Normalmente MyISAM).

Após criar uma tabela, o usuário pode incluir um registro através de uma instrução DML semelhante ao Quadro 15. Assim que a tabela estiver populada com dados é possível alterar (Quadro 16) e excluir registros (Quadro 17). Para excluir uma tabela deve ser executada uma instrução como a do Quadro 18. No Quadro 19 há um exemplo de uma instrução DML para consultar algum registro. Executando esta consulta após a criação da tabela e a inserção do dado apresentado nos exemplos, tem-se o resultado apresentado na Figura 22.

|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | **CREATE TABLE** Vendedor (  Codigo **INT**,  Descricao **VARCHAR(50)**  ) **TYPE** = Vogal |   Quadro - DDL de criação de tabela do tipo Vogal | |  | | --- | | **INSERT INTO** Vendedor  (Codigo**,** Descricao)  **VALUES** (10, ‘José’) |   Quadro - DML de inserção de registro |
| |  | | --- | | **UPDATE** Vendedor  **SET** Descricao = ‘João’  **WHERE** Codigo **=** 10 |   Quadro - DML de alteração de registro | |  | | --- | | **DELETE FROM** Vendedor  **WHERE** Codigo **=** 10 |   Quadro - DML de exclusão de registro |
| |  | | --- | | **DROP TABLE** Vendedor; |   Quadro - DDL de remoção de tabela | |  | | --- | | **SELECT \***  **FROM** Vendedor  **WHERE** Codigo **=** 10 |   Quadro - DML de consulta de registro |

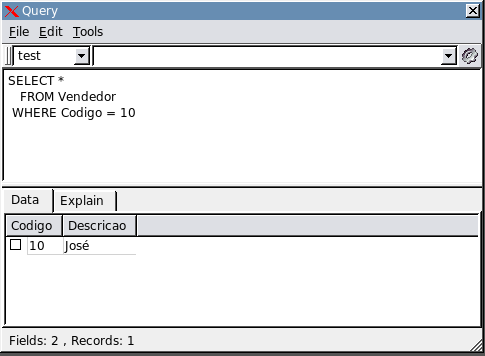


Figura - Consulta e resposta do banco de dados

## RESULTADOS E DISCUSSÃO

O Tipo de Tabela Vogal foi construído com o intuito de diminuir a redundância e aumentar a performance em consultas pontuais. Essa implementação trouxe diversos problemas, dentre eles a indexação Árvore B+ pois inicialmente foi escolhido este tipo de indexação para as colunas. Porém, por definição, este tipo de indexação gera informação redundante. Dessa forma, foi utilizado o tipo de indexação Árvore B normal. Este trás mesmo grau de balanceamento, porém com mais desperdício de espaço nos blocos utilizados, contudo, sem redundância. Sendo assim possível otimizar no futuro.

Outro problema foi de atualização de ponteiros, pois no conceito do projeto, as colunas são como índices independentes, mas esses índices independentes devem estar ligados de alguma forma uns aos outros para possibilitarem a formação de um registro. Dessa forma, uma tabela, além dos índices correspondentes às colunas, há um índice correspondente aos RIDs, ou seja, as colunas apontam para um RID e o RID aponta para as colunas. Como o RID poderia apontar para as colunas? Caso as colunas também tivessem uma numeração sequencial, seriam necessários dois índices para manter uma só coluna, o que tornou essa metodologia inviável. A melhor forma encontrada foi o RID apontar para a posição física do nó da árvore do índice, ou seja, seu bloco e deslocamento. Esta metodologia permite um acesso direto aos dados da coluna em caso de não utilização desta em nenhuma busca. Porém traz um problema bastante grave: este deslocamento pode ser atualizado para cada atualização da coluna, ou seja, se um bloco possuir dados de vinte registros, os RIDs desses registros devem ter o deslocamento referente às colunas modificadas atualizado, e é bastante comum que estes RIDs estejam em blocos independentes. Portanto, segundo o exemplo, para a atualização de um registro, pode ser necessário gravar vinte e um blocos. Os testes mais adiante trarão resultados mais claros.

O gerenciamento de memória também trouxe problemas, pois, ao contrário do que se imaginava inicialmente, o MySQL não faz este gerenciamento para os *storage engines*. Estes devem implementar um gerenciamento individual. Contudo, para uma completa otimização de um sistema, o gerenciamento de memória não pode ser ignorado, e este é um dos grandes segredos dos grandes bancos de dados. Portanto, não há tempo hábil para criação de um gerenciamento de memória tão eficiente quanto às grandes construções (InnoDB, MyISAM, entre outros), mesmo estas sendo de código aberto, pois estas estruturas estão fortemente acopladas à arquitetura de escrita e leitura. Em virtude disso, o gerenciamento de memória é mínimo, procurando otimizar os mecanismos mais simples, como a obtenção de blocos livres.

A documentação disponível no site do MySQL não deixa claro se este é desenvolvido em C ou C++. Apenas no momento que é lida a documentação que vem junto ao código fonte do sistema que é destacada a informação que o MySQL é desenvolvido em C++ e é orientado a objetos. Também é possível constatar isso estudando o código fonte. Em virtude disso o protótipo foi desenvolvido em C em programação estruturada. Para não perder o que foi desenvolvido no protótipo, parte da programação estruturada foi mantida, ou seja, a programação do *storage engine* não é completamente orientada a objetos em virtude do tempo disponível para seu desenvolvimento.

Em consequência aos problemas encontrados e do tempo disponível, foi necessário estabelecer algumas limitações ao projeto. Algumas destas limitações são comparadas ao tipo de tabela Maria no Quadro 20.

|  |  |  |
| --- | --- | --- |
| Tipo de tabela  Limitações | Maria | Vogal |
| controle transacional | Possui | Não possui |
| integridade referencial | Possui | Não possui |
| ACID[[14]](#footnote-15) | Não possui | Não possui |
| tamanho máximo do banco de dados | 256TB | 100MB |
| tamanho máximo de cada registro | 64KB | 1KB |
| máximo de colunas por tabela | 4096 | 1024 |
| tamanho máximo de campos LOB | 4GB | Não possui |
| tamanho máximo de campos de texto | 64KB | 127B |
| tamanho máximo de campos numéricos | 64 *bits* | 32 *bits* |
| menor data válida | 1000 | Não possui |
| maior data válida | 9999 | Não possui |

Quadro - Comparativo de limitações dos tipos de tabela Vogal e Maria

A arquitetura de gravação dos campos torna-os virtualmente ilimitados em uma tabela do tipo Vogal. Tanto o tamanho dos registros, quanto a quantidade de registros ou o tamanho máximo do banco de dados. Porém, as limitações são aqui destacadas até o ponto onde puderam ser testadas e confirmadas.

# DESENVOLVIMENTO DA PROVA DE CONCEITO

Neste capítulo é apresentado o desenvolvimento da prova de conceito que compreende o levantamento dos requisitos da prova, sua implementação e os resultados obtidos.

## requisitos principais do problema a ser trabalhado

A prova de conceito deverá traçar uma análise comparativa entre o tipo de tabela proposto e os tipos InnoDB e MyISAM do MySQL (RF).

## ESPECIFICAÇÃO

Para atender o requisito do projeto é necessário definir o que é relevante ser testado e a que resultados chegar. Baseando-se nas comparações de NG (2009), inicialmente é necessário delinear claramente quais são as limitações e quais são as capacidades dos tipos de tabela testados. Como limitações serão verificados os seguintes itens:

1. tamanho máximo do banco de dados;
2. tamanho máximo de cada registro;
3. máximo de colunas por tabela;
4. tamanho máximo de campos *Large Object* (LOB);
5. tamanho máximo de campos de texto;
6. tamanho máximo de campos numéricos;
7. menor data válida;
8. maior data válida.

Serão analisados os seguintes itens considerados como capacidades do tipo de tabela apenas itens que não são considerados básicos (Exemplos: consulta, inserção, exclusão, atualização, entre outros):

1. união e diferença (o banco de dados MySQL não implementa intersecção);
2. junção interna;
3. junção externa;
4. consulta interna (*sub-select*);
5. LOBs;
6. domínio;
7. programa;
8. ACID;
9. integridade referencial;
10. controle transacional.

Embora as limitações e capacidades sejam importantes para escalar um tipo de tabela, o foco desta prova de conceito é a performance do mesmo. Para tal, baseado nos testes realizados por Pires, Nascimento e Salgado (2008), serão feitos testes de carga, estrutura e consulta. Este teste será realizado com volume de cem mil registros. Como testes de carga e estrutura serão realizados as seguintes verificações:

1. criação de tabela;
2. carregamento da mesma com dados;
3. atualização de um registro;
4. atualização de 100% dos registros;
5. exclusão de um registro;
6. exclusão de 100% dos registros.

Como testes de consulta serão realizados as seguintes validações:

1. consulta de 100% dos registros com projeção total das colunas;
2. consulta de 100% dos registros com projeção parcial (metade das colunas);
3. consulta de 10% dos registros dos registros com projeção total das colunas;
4. consulta de 10% dos registros com projeção parcial;
5. consulta de um registro com projeção total das colunas;
6. consulta de um registro com projeção parcial.

Todos os testes serão realizados em um servidor MySQL local instalado em uma máquina virtual com o sistema operacional Linux na distribuição Ubuntu. Esta máquina virtual é executada reservando 320MB de memória e 20GB de disco rígido em um computador com processador Intel Dual Core T2300 @ 1.66GHz com 1GB de memória RAM DDR2 533 com sistema operacional Microsoft Windows XP Professional Service Pack 3.

## IMPLEMENTAÇÃO

As limitações e capacidades dos tipos de tabela InnoDB e MyISAM são obtidas através da documentação disponível no manual de referência do MySQL de autoria de Dubois (2008) ou através de testes diretos na base de dados.

O OSDB já possui uma implementação bastante completa para testes sobre o MySQL. Portanto, o Quadro 20 demonstra um exemplo parcial do resultado de um teste do OSBC com mil registros, ou seja, para executar os testes de carga, estrutura e consulta, serão necessárias duas iterações (com mil e cem mil registros) para cada tipo de tabela testado.

|  |
| --- |
| OSDB 0.21  Invoked: osdb-my --mysql=innodb --nomulti --noindexes --restrict MySQL  Compile-time options: none  Restrictions: insert\_into rollback subqueries views  create\_tables() 0.07 seconds return value = 0  load() 0.07 seconds return value = 0  Logical database size 0.4000MB (0.3815MiB, 1000 tuples/relation)  join\_3\_cl() 0.01 seconds return value = 0  sel\_10pct\_ncl() 0.01 seconds return value = 0  join\_2\_cl() 0.01 seconds return value = 0  join\_4\_cl() 0.01 seconds return value = 0  proj\_100() 0.01 seconds return value = 966  proj\_10pct() 0.01 seconds return value = 100  bulk\_save() 0.04 seconds return value = 0  bulk\_modify() 0.01 seconds return value = 0  upd\_mod\_t\_mid() 0.01 seconds return value = 0  upd\_mod\_t\_end() 0.02 seconds return value = 0  upd\_mod\_t\_cod() 0.01 seconds return value = 0  bulk\_append() 0.01 seconds return value = 0  Single User Test 0.20 seconds (0:00:00.20) |

Quadro - Exemplo execução OSDB

No Quadro 22 é feito o mapeamento das funções de teste do OSBD com os resultados definidos na especificação. Contudo, houve a necessidade de criar funções específicas no OSBD para que fosse possível executar os testes necessários para obter os resultados previstos.

|  |  |  |
| --- | --- | --- |
|  | Resultado esperado | Função OSDB |
| Carga e estrutura | criação das tabelas | create\_tables |
| carga de dados | load |
| atualização de um registro | upd\_unique |
| atualização de 100% dos registros | upd\_100pct |
| exclusão de um registro | del\_unique |
| exclusão de 100% dos registros | del\_100pct |
| Consulta | 100% dos registros (projeção total) | selproj\_100\_100pct |
| 100% dos registros (projeção parcial) | selproj\_100\_50pct |
| 10% dos registros (projeção total) | selproj\_10\_100pct |
| 10% dos registros (projeção parcial) | selproj\_10\_50pct |
| um registro (projeção total) | selproj\_1\_100pct |
| um registro (projeção parcial) | selproj\_1\_50pct |

Quadro - Mapeamento dos resultados esperados com as funções do OSDB

### Técnicas e ferramentas utilizadas

O OSDB foi utilizado como ferramenta de teste da implementação sobre a mesma plataforma configurada para o ambiente de desenvolvimento do Tipo de Tabela.

#### Código desenvolvido

Para criar um determinado volume de dados deve ser executado o comando osdb-my --generate-files --size 40mb --datadir /tmp onde o parâmetro size corresponde ao volume de dados criado para os testes. Para criar um volume de cem mil linhas é necessário um tamanho de aproximadamente 40MB. Depois de criado um determinado volume é iniciado a execução do teste a partir do comando osdb-my --mysql=innodb --nomulti --noindexes --restrict MySQL. Onde o parâmetro mysql identifica em que *storage engine* as tabelas serão criadas, o parâmetro nomulti indica que o teste não será executado simulando múltiplos usuários e o parâmetro noindexes desativa a criação de índices. O parâmetro restrict MySQL não executa os testes que contenham instruções INSERT INTO, ROLLBACK, VIEW ou *subqueries*. Essa parametrização é importante para equilibrar as características dos tipos de tabelas para um teste mais próximo da realidade, diminuindo as diferenças causadas por otimizações que não existam no Tipo de Tabela Vogal.

## RESULTADOS E DISCUSSÃO

O tipo de tabela Vogal possui limitações e capacidades definidas de acordo com este projeto. As limitações dos tipos de tabela são apresentadas no Quadro 23 e as capacidades dos tipos de tabela são apresentadas no Quadro 24.

|  |  |  |  |
| --- | --- | --- | --- |
| Tipo de tabela  Limitações | InnoDB | MyISAM | Vogal |
| tamanho máximo do banco de dados | 64TB | 256TB | 100MB |
| tamanho máximo de cada registro | 8KB | 64KB | 1KB |
| máximo de colunas por tabela | 1000 | 4096 | 1024 |
| tamanho máximo de campos LOB | 4GB | | Não possui |
| tamanho máximo de campos de texto | 64KB | | 127B |
| tamanho máximo de campos numéricos | 64 *bits* | | 32 *bits* |
| menor data válida | 1000 | | Não possui |
| maior data válida | 9999 | | Não possui |

Quadro - Limitações dos tipos de tabela

|  |  |  |  |
| --- | --- | --- | --- |
| Tipo de tabela  Capacidades | InnoDB | MyISAM | Vogal |
| união, diferença | Sim | Sim | Não |
| junção interna |
| junção externa |
| consulta interna (*sub-select*) |
| LOBs |
| domínio |
| programa |
| ACID |
| integridade referencial |
| controle transacional |

Quadro - Capacidades dos tipos de tabela

O resultado dos testes de carga, estrutura e consulta com um volume de cem mil registros são destacados na Quadro 25. Estes valores são a média de cinco execuções e estão representados em segundos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Tipos de tabela  Testes | InnoDB | MyISAM | Vogal |
| Carga e estrutura | criação das tabelas | 0,06 | 0,03 |  |
| carga de dados | 19,16 | 4,77 |  |
| atualização de um registro | 0,32 | 0,81 |  |
| atualização de 100% dos registros | 2,69 | 2,65 |  |
| exclusão de um registro | 1,44 | 0,12 |  |
| exclusão de 100% dos registros | 1,75 | 0,01 |  |
| Consulta | 100% dos registros (projeção total) | 2,26 | 1,91 |  |
| 100% dos registros (projeção parcial) | 1,25 | 1,14 |  |
| 10% dos registros (projeção total) | 0,52 | 0,37 |  |
| 10% dos registros (projeção parcial) | 0,36 | 0,25 |  |
| um registro (projeção total) | 0,34 | 0,13 |  |
| um registro (projeção parcial) | 0,70 | 0,14 |  |

Quadro - Resultados dos testes com cem mil registros

# CONCLUSÕES

Para elaborar a conclusão é necessário concluir os testes da prova de conceito.

[Deve-se também incluir aqui as principais vantagens do seu trabalho e limitações.]

O tipo de tabela proposto, como módulo funcional do SGBD MySQL, é uma otimização das estruturas de dados existentes visando remover definitivamente das mãos do desenvolvedor a responsabilidade de equilibrar a normalização com o desempenho, ou seja, o processo de normalização além de ser viável seria altamente aconselhado para uma melhor eficiência, sendo eficiência uma relação entre performance e uso de recursos. O tipo de tabela Maria do MySQL, apresentado no tópico 2.5.1, segue um propósito semelhante ao tentar aprimorar os mecanismos de segurança do tipo de tabela MyISAM com o intuito de substituí-lo.

De forma superficial, é no registro que está a mudança mais drástica de paradigma proposta por este trabalho. Ele não será mais uma estrutura física direta e linear, e sim sua estrutura física será uma relação de dados gravados separadamente para formar um registro apenas logicamente linear. Sendo assim, a estrutura a ser desenvolvida necessita de uma quantidade de ponteiros relativamente maior que as estruturas existentes, e isso é importante para reduzir redundância, reforçar o inter-relacionamento, potencialmente tornando mais rápidas uma ampla variedade de consultas. Esta arquitetura tende a melhorar o desempenho de consultas pontuais (que tem uma cláusula de condição mais restritiva, ou seja, retornam relativamente poucos registros) sem criar quaisquer índices, pois, ao invés de buscá-los para tentar otimizar a consulta, cada coluna seria como um índice por si só, não havendo necessidade de estruturas adicionais. O índice do tipo Bitmap (tópico 2.5.2) também visa otimizar a obtenção de informação. Contudo, tem como propósito melhorar o desempenho de consultas mais abrangentes (que retornam uma grande quantidade de registros).

Portanto, existe um grande potencial de ganho de performance e redução de volume de armazenamento. Porém, apenas aplicando numerosos testes e simulando o uso real pode-se chegar a uma conclusão mais consistente. Atinge-se esta conclusão realizando um benchmarking como prova de conceito, tendo como base o comparativo de desempenho entre bancos de dados de código aberto realizado por Pires, Nascimento e Salgado (2006).

Uma vez estável, este tipo de tabela será disponibilizado para a comunidade de código aberto junto ao MySQL, esperando desta forma a contribuir e receber contribuições de quem se mostrar interessado.

## EXTENSÕES

São possíveis inúmeras extensões a este projeto, dentre elas:

1. ultrapassar os limites do Tipo de Tabela definidos neste projeto para garantir que as estruturar possibilitem uma faixa de limite muito mais abrangente;
2. permitir os tipos de campo de ponto flutuante, data, LOBs, entre outros;
3. implementar um gerenciamento de memória mais robusto;
4. desenvolver um gerenciamento de transações (Se possível ACID);
5. possibilitar junções (externas, internas, naturais, entre outras);
6. adicionar funções de conjunto como união e interseção;
7. incluir funções de agrupamento: group by, sum, avg, distinct, entre outros;
8. validar obrigatoriedade e limites de campos;
9. aproveitar mecanismo de incremento automático no valor das colunas presente no MySQL;
10. implementar seleção por faixa (LIMIT);
11. configurar o arquivo de dados, seu tamanho, e adicionar possibilidade de que ele aumente automaticamente;
12. desenvolver mecanismo de rebalanceamento das árvores de índices;
13. implementar mecanismo de coleta de estatísticas dos índices para possibilitar a montagem de um plano de execução mais apurado;
14. compatibilizar com todos os sistemas operacionais em que o MySQL é compatível além do Linux;
15. melhorar o tratamento de erros para utilizar os mesmos padrões do desenvolvimento do MySQL aproveitando sua infraestrutura de tratamento de exceções;
16. construir mecanismo de visões para possibilitar a consulta das tabelas do metadados.

Referências bibliográficas

CROOK, Neal. **Brief introduction to alpha systems and processors**. [S.l.]: Digital Equipment, 1997. Disponível em: <http://www.alphalinux.org/docs/alpha-howto.html>. Acesso em: 23 out. 2008.

CYRAN, Michele et al. **Oracle® database**: concepts - 10g release 2. [S.l.], 2005. Disponível em: <http://download.oracle.com/docs/cd/B19306\_01/server.102/b14220/toc.htm>. Acesso em: 20 out. 2008.

DUBOIS, Paul et al. **MySQL 5.1 reference manual**. [S.l.], 2008. Disponível em: <http://dev.mysql.com/doc/refman/5.1/en>. Acesso em: 20 out. 2008.

ELMASRI, Rasmez; NAVATHE, Shamkant B**. Sistemas de bancos de dados**. 4.ed. Revisor Técnico Luis Ricardo de Figueiredo. São Paulo: Pearson Addison Wesley, 2005.

GARCIA-MOLINA, Hector; ULLMAN, Jeffrey D.; WIDOM, Jennifer: **Implementação de sistemas de bancos de dados**. Tradução Vanderberg D. de Souza. Rio de Janeiro: Campus, 2001.

HEINZLE, Roberto. **Estrutura de dados**: implementações com C e Pascal. Blumenau: Diretiva, 2006.

MYSQL AB. **Why MySQL?** [S.l.], 2008. Disponível em: <http://www.mysql.com/why-mysql/>. Acesso em: 22 out. 2008.

MYSQL COMMUNITY. **MySQL internals**. [S.l.], 2008. Disponível em: <http://dev.mysql.com/doc/internals/en>. Acesso em: 20 out. 2008.

NG, Martin et al. **Comparison of relational database management systems**. [S.l.], 2009. Disponível em: <http://en.wikipedia.org/wiki/Comparison\_of\_relational\_database\_management\_systems>. Acesso em: 01 jun. 2009.

PIRES, Carlos E. D.; NASCIMENTO, Rilson O.; SALGADO, Ana C. Comparativo de desempenho entre bancos de dados de código aberto. In: ESCOLA REGIONAL DE BANCO DE DADOS. 2., 2006, Passo Fundo. **Anais**... Porto Alegre: Sociedade Brasileira de Computação, 2006. p. 21-26. Disponível em: <http://www.upf.br/erbd/download/15997.pdf> Acesso em: 30 out. 2008.

RAMAKRISHNAN, Raghu. **Database management systems**. Singapura: WCB McGraw-Hill, 1998.

RIEBS, Andy. **OSDB readme**. [S.l.], 2004. Disponível em: <http://osdb.cvs.sourceforge.net/viewvc/osdb/osdb/README?revision=1.8&view=markup>. Acesso em: 30 mai. 2009.

TOKUTEK. **Performance brief**: TokuDB for MySQL. [S.l.], 2009. Disponível em: <http://www.tokutek.com/mysql-performance-brief.pdf>. Acesso em: 30 maio 2009.

APÊNDICE A – Diagrama de classe completo

O diagrama de classes completo (com atributos e métodos) do Tipo de Tabelaé destacado na Figura 23.



Figura - Diagrama de classes completo do Tipo de Tabela

1. Tipo de tabela é um recurso disponibilizado pelo SGBD MySQL que permite definir estruturas de dados diferentes para cada tabela criada na base de dados mantendo a possibilidade de relacionamento. Cada tipo de tabela é mantida por um *storage engine* o qual funciona como uma camada de persistência que é ligada às camadas de controle do banco de dados. A partir deste ponto do trabalho será referenciado como Tipo de Tabela. [↑](#footnote-ref-2)
2. MySQL é um SGBD de código aberto e portável (DUBOIS, 2008). [↑](#footnote-ref-3)
3. InnoDB é um tipo de tabela do MySQL que implementa controle transacional, integridade referencial e bloqueio de alterações no nível de registro (DUBOIS, 2008). [↑](#footnote-ref-4)
4. MyISAM é um tipo de tabela do MySQL baseada no formato ISAM (Indexed Sequential Access Method) da IBM© e otimizada para ser especialmente veloz no processo de consulta (DUBOIS, 2008). [↑](#footnote-ref-5)
5. *Structured Query Language* (SQL) é uma linguagem formal baseada na álgebra relacional que se aproxima da linguagem humana com o intuito de facilitar a interação homem-máquina. É amplamente utilizada por SGBDs como meio de comunicação do usuário. [↑](#footnote-ref-6)
6. MySQL AB foi comprada pela Sun Microsystems em fevereiro de 2008 e a Sun, em abril de 2009, foi comprada pela Oracle. [↑](#footnote-ref-7)
7. Recompilar é o ato de compilar mais que uma vez. Compilar é todo o processo desde a validação dos códigos da linguagem de programação até sua tradução para a linguagem de máquina. [↑](#footnote-ref-8)
8. GNU Linux/Alpha é uma versão do GNU Linux que funciona sobre a arquitetura Alpha criada pela Compaq. Alpha é o nome dado para o projeto de processadores com a arquitetura *Reduced Instruction Set Computer* (RISC) de 64 bits criada pela empresa norte americana Digital Equipment (CROOK, 1997). [↑](#footnote-ref-9)
9. Usuário, no contexto desta monografia, significa o ator que utilizará o SGBD. Na maioria dos casos este ator é outro sistema. [↑](#footnote-ref-10)
10. Metadados é a definição básica de um esquema em um banco de dados, ou seja, é a estrutura primordial que identifica os objetos existentes e orienta o SGDB. [↑](#footnote-ref-11)
11. Cliente, no paradigma Cliente/Servidor, significa uma aplicação que envia solicitações para um servidor, que processa a solicitação, gera uma resposta e a envia ao cliente. O MySQL é um banco de dados cliente/servidor. Portanto, cliente neste contexto representa uma aplicação que se comunica com o servidor através de uma linguagem comum, no caso: SQL. [↑](#footnote-ref-12)
12. DDL é um subconjunto de instruções SQL que tem como característica a definição/alteração da esquema de dados. [↑](#footnote-ref-13)
13. DML é um subconjunto de instruções SQL que tem como característica a manipulação de dados em uma tabela existente no esquema de dados. [↑](#footnote-ref-14)
14. ACID é acrônimo de *Atomic, Consistency, Isolation, Durability*, cuja tradução é Atômico, Consitente, Isolado e Durável. Transações implementadas de forma apropriada satisfazem as características ACID sendo que, para uma transação ser atômica significa que é na base de tudo ou nada; Para ser isolada, a transação deve ser executada como se nenhuma outra transação estivesse em execução ao mesmo tempo; Tem a característica de durável a transação que sua conclusão garante a persistência da informação no banco de dados; Para ser consistente a informação não pode sofrer mudanças sem que seja sobre ação direta do usuário (GARCIA-MOLINA; ULLMAN; WIDOM, 2001, p. 10). [↑](#footnote-ref-15)