UNIVERSIDADE DO ESTADO DA BAHIA
DEPARTAMENTO DE CIÊNCIAS EXATAS E DA TERRA
BACHARELADO EM ANÁLISE DE SISTEMAS

CONSTRUÇÃO DE UM PROTÓTIPO PARA INDEXAR DOCUMENTOS DE TEXTO LONGO DIRECIONADO AO FIREBIRD

Edmilson dos Santos de Jesus

Salvador- BA
Agosto 2003

UNIVERSIDADE DO ESTADO DA BAHIA
DEPARTAMENTO DE CIÊNCIAS EXATAS E DA TERRA
BACHARELADO EM ANÁLISE DE SISTEMAS

CONSTRUÇÃO DE UM PROTÓTIPO PARA INDEXAR DOCUMENTOS DE TEXTO LONGO DIRECIONADO AO FIREBIRD

Edmilson dos Santos de Jesus

Monografia apresentada ao Curso de Análise de Sistemas da Universidade do Estado da Bahia como requisito para obtenção dos créditos na disciplina de Projeto Final obrigatória para obtenção do título de Bacharel em Análise de Sistemas.

Orientador : Jorge Sampaio Farias

Salvador- BA
Agosto 2003

Dedico esse trabalho aos meus pais José e Maria que se esforçaram ao máximo para que eu tivesse uma boa educação e que sempre estiveram perto nos momentos mais difíceis.

AGRADECIMENTOS

Quero agradecer a Deus por ter me dado a oportunidade de estar aqui, entre vós, e poder contribuir de alguma forma para o desenvolvimento de nossa sociedade.

Ao professor Jorge Sampaio Farias pela orientação indispensável, pelas longas discussões que tivemos ao longo do projeto, pela amizade e pela confiança em mim depositados.

A Carlos Eduardo Martins Conceição, por ter permitido que eu me apossasse de seu livro de cabeceira sobre banco de dados, pela amizade e pelas longas discussões que tivemos no OOLAB sobre o trabalho.

A Eduardo Rocha Rodrigues, colaborador permanente do OOLAB, pela amizade e apoio e incentivo desde o longo período em que trabalhamos juntos até os dias de hoje.

Aos veteranos do OOLAB que contribuíram diretamente no meu aprendizado durante a longa jornada que estivemos juntos dentro e fora da universidade.

A Albert e todos os integrantes da nova geração do OOLAB, que contribuíram no desenvolvimento desse trabalho.

A Ana América, secretária do Colegiado de Análise de Sistemas, pela amizade, paciência, orientação e contribuição concedidas desde os primeiros semestres.

A todos aqueles que não mencionei seus nomes mas que de alguma forma contribuíram para a realização deste trabalho.

RESUMO

Nos últimos anos muitos estudos vêem sendo desenvolvidos na área de indexação de documentos de texto, porém a maioria das estruturas propostas requerem sofisticados recursos de hardware para seu processamento. Além disso são raros os estudos nessa área cuja implementação esteja voltada para Sistemas de Gerenciamento de Bancos de Dados.

Atualmente os recursos disponíveis para a construção de índices de documentos em bancos de dados se limitam a ferramentas externas ao SGBD que precisam ser instaladas e configuradas para tal finalidade. Exigindo dos desenvolvedores conhecimentos específicos sobre a ferramenta utilizada. Além disso essas ferramentas criam e gerenciam sua própria estrutura de índices, ou seja, os índices criados são armazenados fora do SGBD.

Esse trabalho apresenta uma proposta de estrutura de índice, para arquivos de texto longo, preparada para ser armazenada em páginas do SGBD. Um protótipo do modelo proposto foi implementado e sua aplicação direcionada para textos armazenados em campos do tipo BLOB no Firebird. São apresentadas duas técnicas utilizadas na busca por frases em bancos de dados textuais. O índice proposto é implementado utilizando-se uma delas, a técnica de índice invertido em árvores B+. Apresenta-se, ainda, considerações sobre a arquitetura interna do Firebird e seu funcionamento. Alguns aspectos da arquitetura e implementação do índice utilizado pelo famoso mecanismo de busca Google também são discutidos.

Palavras Chave : BLOB, índice invertido, índice para a próxima palavra, Firebird, árvore B+, índice bitmap, busca por frases, Google, PageRank.

LISTA DE FIGURAS

LISTA DE SIGLAS E ABREVIATURAS

1. SGBD (Sistema de Gerenciamento de Bancos de Dados) – Sistema responsável pelo armazenamento, organização e gerenciamento de dados, garantindo a consistência dos mesmos nele armazenados.
2. SGBDR (Sistema de Gerenciamento de Banco de Dados Relacional) – SGBD onde os dados são representados em linhas e colunas.
3. B-Tree – Abreviação utilizada para árvore B.
4. InterBase – SGBD Relacional desenvolvido e comercializado pela Borland.
5. CVS - Sistema de Controle de Versões, utilizado para gerenciar desenvolvimento de aplicações.
6. Classic – Refere-se ao primeiro modelo de arquitetura implementado no Firebird.
7. Cache – Memória utilizada para armazenar dados temporariamente.
8. Checksum – Número de checagem, utilizado para validar a integridade de arquivos.
9. Firebird – SGBD relacional de código fonte aberto criado a partir do InterBase.
10. Threads –São linhas de execução que pertencem a uma mesma aplicação e que concorrem pelo uso do processador, permitindo o uso mais efetivo do mesmo.
11. Clustered Index – É um índice onde os dados estão fisicamente ordenados e a as páginas de dados são as folhas do índice.
12. DLL (Dynamic Link Library) – Biblioteca de acesso dinâmico, usada por aplicativos para utilizar recursos externos aos mesmos.
13. BLOB (Binary Large Object) – Objeto binário extenso, capaz de armazenar grandes quantidades de dados.
14. Drive – Refere-se ao conjunto de rotinas que permitem o acesso ao SGBD para manipulação de um banco de dados. Essas rotinas são específicas para cada SGBD, e são definidas pelo fabricante.
15. Metadados – Catálogo de informações sobre os dados armazenados no banco de dados.
16. ODS (On Disk Structure) – Refere-se a estrutura de dados em disco utilizada pelos arquivos do Firebird.
17. SO (Sistema Operacional) – Sistema que gerencia os recursos do computador.
18. URL (Localizador Uniforme de Recursos)- Um endereço de uma página na Web.
19. KB (Kbyte) – Utilizada para expressar quantidades de bytes, 1024 bytes eqüivale a 1KB.
20. MB (Megabyte) – Utilizada para expressar quantidades de bytes, onde 1024 Kbytes eqüivale a 1 MB.

21. GB (Gigabyte) – Utilizada para expressar quantidades de bytes, onde 1024 Mbytes eqüivale a 1 GB.
22. PC (Computador Pessoal) – Microcomputador de uso pessoal.

LISTA DE QUADROS

LISTA DE GRÁFICOS

MARCAS E PATENTES

Microsoft SQL Server e Microsoft Search Service são marcas registradas da Microsoft Corporation.

Interbase, Delphi e C++Builder são marcas registrada da Borland Corporation. PageRank e Google são marcas registradas da Google Inc.

Oracle é marca registrada da Oracle Corporation.

SUMÁRIO

1. INTRODUÇÃO

Segundo Garcia-Molina et. al. (2001, p.1) “Os bancos de dados são essenciais para todos os ramos de negócios”, e não é muito difícil notar isso. Com a evolução tecnológica, a cada dia aumenta a capacidade de armazenamento e processamento dos computadores. Isso aliado a crescente utilização dos computadores na sociedade, tem alavancado o crescimento da utilização dos bancos de dados em todas as áreas do conhecimento, inclusive nos campos mais inusitados. E os Sistemas de Gerenciamento de Bancos de dados (SGBD’s) também acompanham essa evolução, estando a cada dia mais robustos e aptos a armazenar grandes quantidades de dados, chegando a casa dos terabytes.

Para permitir a flexibilidade dada a sua capacidade de armazenamento os SGBD’s suportam muitos campos de tamanho variável como varchar() (texto de tamanho variável) e BLOB (Binary Large Object) .

Os campos do tipo BLOB, podem armazenar, atualmente, até gigabytes de texto, imagem ou vídeo, e é cada dia mais comum a utilização desse tipo de campo para armazenar grandes quantidades de dados. Entretanto, buscar por valores nesses campos é uma tarefa lenta e evitada por muitos desenvolvedores de sistemas, visto que os SGBD’s atuais, não permitem a indexação direta desses campos, exigindo do desenvolvedor uma habilidade extra na manipulação de ferramentas específicas incorporadas ao banco que auxiliam a busca nesses campos.

1.1. OBJETIVOS

Sabendo-se que um campo BLOB tem tamanho variável e pode armazenar até 32 gigabytes de dados, de acordo com o tamanho de página do banco de dados, a proposta desse projeto é investigar e implementar um método capaz de indexar arquivos de texto longo armazenados nesse tipo de campo. Além disso busca-se alcançar com esse trabalho os seguintes objetivos específicos:

(1) Avaliar os métodos disponíveis para indexação de textos longos em campos to tipo BLOB.

(2) Levantar os caminhos disponíveis para construção de uma estrutura de índices que comporte grandes quantidades de dados.

(3) Especificar e implementar um protótipo que comporte indexação de arquivos de texto longo.

(4) Determinar uma forma de mensuração de resultados e implementá-la.

(5) Investigar e apontar as modificações necessárias no SGBD Firebird para comportar a estrutura de índice proposta.

1.2. JUSTIFICATIVA

Em um SGBD os índices nada mais são que estruturas de dados ordenadas por uma chave formada por um ou mais campos. Essa chave aponta para o endereço onde o registro relacionado àquela chave se encontra no banco de dados. Assim é possível procurar por um dado pesquisando nessa estrutura, sem que seja necessário percorrer seqüencialmente os registros armazenados no banco.

Entretanto a eficiência de um índice está, de certa forma, relacionada a composição de sua chave, de tal modo que um índice composto por uma chave muito grande, seja ela formada por muitos campos ou com campos muito grandes, pode degradar a performance do SGBD. O uso de um índice que tenha em sua chave campos de tamanho fixo e com alta cardinalidade pode elevar significativamente a performance da consulta. Não é possível se utilizar um campo do tipo BLOB como chave de um índice devido ao tamanho que estes campos podem alcançar. Isso não quer dizer que não seja possível indexá-los.

Assim, como a busca por um dado em um banco é normalmente um processo lento e os índices tendem a otimizar a busca, seria então desejável a utilização desses índices para buscas de texto em campos do tipo BLOB já que estes são os campos que têm uma maior capacidade de armazenamento.

A indexação de campos do tipo BLOB pelo próprio SGBD evitará a utilização de ferramentas externas para auxiliar a busca por dados nesses campos. Além disso as ferramentas disponíveis encontradas se limitam a armazenar o índice na parte externa do SGBD e a maioria delas funcionam como um serviço do Sistema Operacional forçando o banco a se comunicar com tal serviço para consultar e atualizar o índice. Com a implementação interna do índice essa etapa de comunicação será eliminada e o índice será armazenado no próprio SGBD.

Com a crescente utilização desse tipo de campo pelos sistemas de informação as melhorias de performance na recuperação dos dados afetariam beneficamente os usuários de tais sistemas.

O Firebird será utilizado para esse projeto por se tratar de um SGBD multiplataforma, maduro, de código fonte aberto e que é amplamente utilizado pela comunidade mundial de informática, em sistemas acadêmicos e comerciais.

1.3. ORGANIZAÇÃO DO TRABALHO

Este trabalho está estruturado em 7 capítulos descritos a seguir:

O primeiro deles apresenta a introdução, com os objetivos almejados e a justificativa para esse trabalho.

O segundo capítulo apresenta, de forma resumida, a fundamentação teórica necessária para a elaboração do trabalho, começando com as considerações sobre trabalhos anteriores, seguindo com um pouco da teoria usada na construção de índices e, finalmente, são apresentadas as técnicas mais comumente utilizadas na construção de índices em bancos de dados de documentos.

O Firebird, sua história, arquitetura interna e características peculiares, aparecem descritos no terceiro capítulo.

Um breve descritivo sobre as ferramentas disponíveis para efetuar a busca em campos do tipo BLOB, bem como suas características, são apresentadas no capítulo quatro.

O funcionamento do mecanismo de busca Google, suas características e alguns aspectos de implementação, são mostrados no quinto capítulo.

O sexto capítulo trata da especificação de um protótipo para indexar campos BLOB, especificamente no Firebird, apresentando os principais requisitos.

O sétimo capítulo descreve um experimento implementando o índice sobre arquivos de texto longo, mostrando a descrição, codificação, testes e avaliação do mesmo.

Por fim, no oitavo capítulo, são apresentadas as conclusões, dificuldades encontradas, limitações do modelo proposto e sugestões para futuros trabalhos.

2. FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os conceitos e fundamentos necessários ao desenvolvimento do trabalho. O entendimento da teoria aqui apresentada é de vital importância para que sejam alcançados os objetivos do trabalho.

2.1.CONSIDERAÇÕES SOBRE TRABALHOS ANTERIORES

Existe uma grande quantidade de trabalhos que tratam sobre a idexação de documentos. Em Brin and Page (1998) é descrita a arquitetura do Google, um mecanismo de busca que indexa bilhões de documentos da internet.

O Google mantém na memória principal uma tabela de vocabulários. Uma função converte a palavra procurada em uma entrada válida na tabela, na qual a respectiva saída aponta para uma estrutura de índice invertido contendo a lista dos documentos onde a palavra aparece. Esses documentos são identificados univocamente e estão seguidos das posições onde a palavra ocorre dentro dos mesmos. Ao final da busca o algoritmo *PageRank* é utilizado para classificar o resultado. O Google também utiliza vários clusters, permitindo o balanceamento de carga e o processamento paralelo de algumas tarefas.

Em Nagarajarao et al. (2002), é apresentada uma estrutura de índice invertido que suporta atualização incremental na qual vários documentos podem ser adicionados ao índice sem a necessidade de reindexação. Além do que, o índice pode ser pesquisado enquanto está sendo atualizado.

Putz (1991), descreve uma forma de implementar índice invertido utilizando um SGBDR (Sistema de Gerenciamento de Banco de Dados Relacional). Nessa implementação o índice é mantido em tabelas do banco e um índice *clustered* é criado no campo que guarda as palavras-chave do índice invertido. No momento da busca esse índice é utilizado.

Essa implementação exige uma modificação do código SQL utilizado para recuperar documentos, além da criação das tabelas e índices.

Atualmente alguns SGBD's como Oracle e SQL Server possuem ferramentas que permitem a criação de índices em campos de texto longo.

O Microsoft SQL Server a partir da versão 7.0 traz um recurso chamado Full-Text Search que permite a criação de um índice externo ao banco que pode ser utilizado para

pesquisa em campos de texto longo. O índice é criado, gerenciado e excluído através de uma lista de procedimentos predefinidos que são distribuídos junto com o SGBD pelo fabricante. Uma vez criado ele é armazenado fora do banco de dados. A estrutura que armazena esses dados é chamada de índice Full-Text.

A tarefa de preencher o índice ou atualizá-lo não é automática e deve ser agendada ou executada manualmente. A razão para isso é que popular um índice desse tipo pode exigir a utilização de muitos recursos computacionais e isso pode degradar a performance do banco.

O índice Full-Text opera como um serviço chamado Microsoft Search e tem como funções principais permitir a indexação através da criação de catálogos e executar as consultas requisitadas pelo SQL Server. Os tipos de consultas suportadas incluem, buscas por palavras, frases e por proximidade de palavras. Além de poder pesquisar pelas diferentes formas de um verbo, ou pelo singular e plural de substantivos.

Internamente o SQL Server envia as condições da pesquisa para o Microsoft Search Service, que por sua vez pesquisa no índice observando essas condições e retornado ao SQL Server as chaves dos registros que atendem aos critérios da busca. O SQL Server então utiliza a lista de chaves para determinar que linhas da tabela devem ser processadas.

Pesquisas direcionadas ao índice Full-Text são usualmente menos precisas do que as pesquisas feitas em índices do próprio banco nos quais o resultado da pesquisa é um conjunto de linhas onde cada linha tem a palavra ou frase pesquisada exatamente da mesma forma como foi escrita. Na verdade utilizando-se o índice Full-Text o resultado da pesquisa pode trazer a forma no plural ou singular de um substantivo pesquisado, as várias formas que um verbo pesquisado pode aparecer no texto, as palavras de uma frase juntas ou separadas classificadas por proximidade, ou pode haver até diferenças entre letras maiúsculas e minúsculas no texto procurado e no encontrado.

Índices Full-Text podem ser utilizados para diferentes propósitos e devem estar sempre atualizados, assim sempre que o dado de uma coluna associada ao índice for modificado o índice Full-Text também deve ser atualizado. No entanto atualizar o índice pode levar tempo, por isso é recomendado pela Microsoft que essas atualizações sejam feitas em momentos de baixa utilização do SQL Server.

2.2.ÍNDICES

Índices são estruturas de dados que são criados para permitir a rápida localização dos registros dentro de uma tabela. Assim, como o índice de um livro ajuda o leitor a encontrar informações sobre determinado assunto mais rapidamente, um índice sobre uma tabela provê uma forma rápida de acessar os dados da mesma.

Sempre que a operação de alteração, exclusão ou modificação de um registro for executada o índice criado sobre tal tabela, contendo o registro em questão, deve ser atualizado. Por esse motivo é que os índices também influenciam na performance de tais operações.

Segundo Garcia-Molina et. al. (2001, p.154) os índices podem ser classificados em primários ou secundários.

Os índices primários são aqueles cujas folhas apontam para o bloco onde estão os registros. Tal índice parte do pressuposto de que os registros estão armazenados em ordem no bloco. Assim só é possível a existência de um único índice primário para cada tabela.

Nos índices considerados secundários, cada folha do índice aponta para o registro correspondente dentro de um bloco, não interessando se os registros estão ou não ordenados.

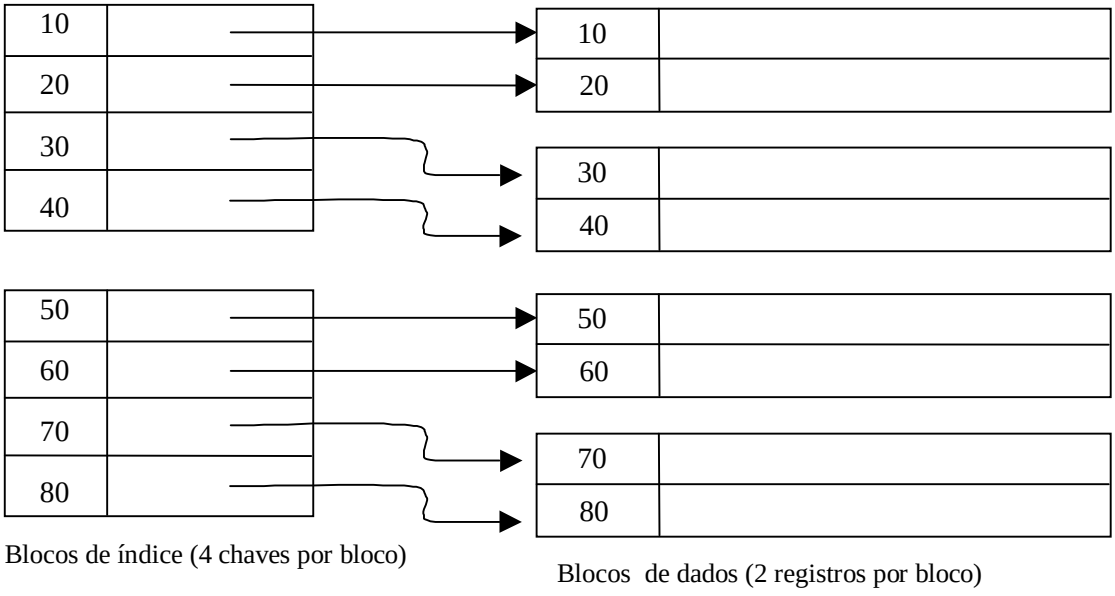


Figura 1 – Índice denso

Existe ainda uma outra classificação que diz respeito a forma como as chaves são distribuídas dentro do índice. Com relação a esse aspecto os índices podem ser densos ou esparsos. Os índices densos são aqueles para os quais existe uma entrada na tabela de índices

para cada entrada na tabela de registros, de forma que para *n* registros existirá sempre *n* entradas na tabela de índices, como mostra a Figura 1.

Em um índice esparso não é necessário uma entrada na tabela de índices para cada registro, porque os dados para os quais ele aponta estão classificados seqüencialmente. Isso permite ao índice armazenar apenas a primeira chave de cada bloco na seqüência, conforme mostra a Figura 2.

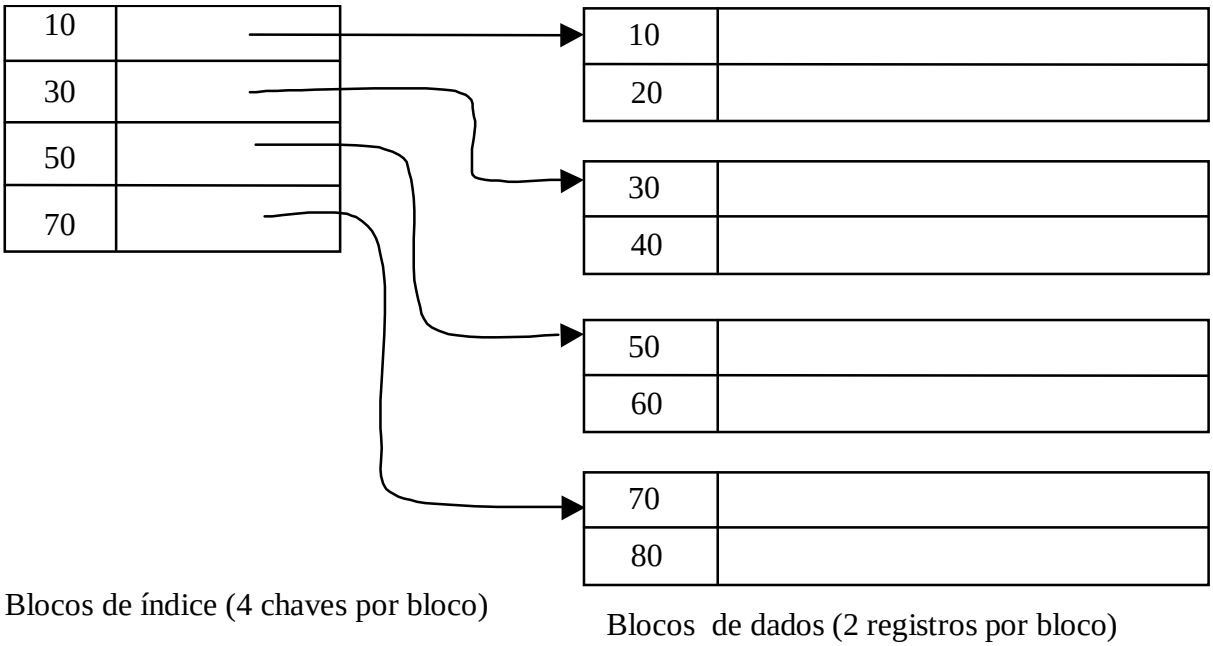


Figura 2 - Índice esparso.

Para tornar a estrutura de índice mas eficiente, é comum a utilização de um nível de índice sobre outro. Assim pode-se construir um índice de *n* níveis, com *n* >=1, onde o primeiro nível, que aponta para os dados, pode ser denso ou esparso e os demais níveis são necessariamente esparsos, pois se forem densos não irão acrescentar qualquer vantagem à estrutura do índice, como cita Garcia-Molina et. al. (2001, p.142), "A razão é que um índice denso sobre um índice teria exatamente tantos pares chave-ponteiro quanto o índice do primeiro nível e, portanto, ocuparia exatamente o mesmo espaço que o índice do primeiro nível."

2.2.1. SELETIVIDADE

A seletividade de um conjunto de uma ou mais colunas é uma medida comum da aplicabilidade de um índice sobre aquelas colunas. Índices ou colunas são seletíveis se têm

um grande número de valores únicos ou se possuem poucos valores duplicados. Assim quanto maior a cardinalidade de uma coluna maior será a seletividade da mesma.

Índices seletíveis são mais eficientes que os não seletíveis pois eles apontam mais diretamente para valores específicos. Portanto um otimizador de consultas baseado em custo tentará utilizar sempre o índice mais seletível.

2.2.2. ÁRVORE B+

Segundo Tenenbaum (1995, p. 554) uma árvore B de ordem *n* é uma árvore de busca multidirecional e balanceada, na qual cada nó não-raiz contém pelo menos *n/2* chaves. Existe uma variação da árvore B chamada árvore B+, que é freqüentemente utilizada na construção de índices para bancos de dados, onde o último ponteiro de cada folha aponta para a primeira chave da próxima folha, como pode ser visto na Figura 3. Em particular essa estrutura é a mais comumente utilizada para a construção de índices. Isso se deve a sua eficiência na busca, além do que os nós da árvore são projetados para que caibam em um bloco, exigindo-se assim, menos operações de entrada/saída para a sua recuperação.

Em muitos casos o índice implementado como *B-tree* pode ser mantido parcialmente na memória principal, o que reduz ainda mais o tempo de busca.

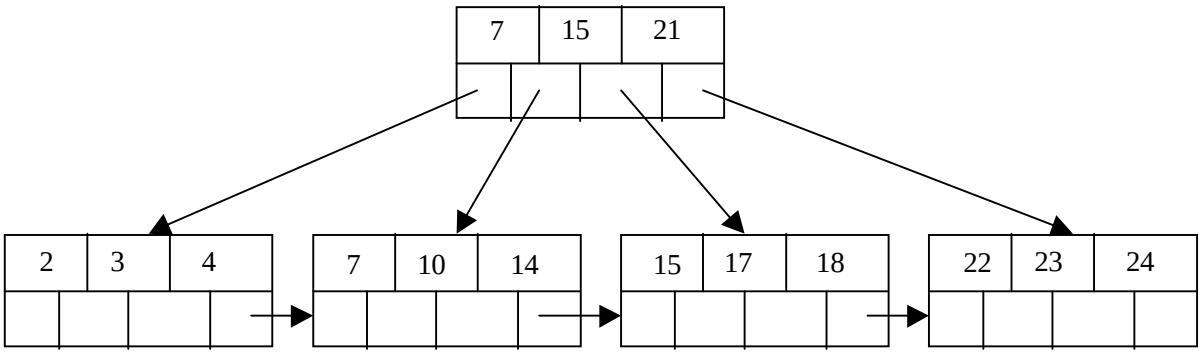


Figura 3 - Árvore B+

2.2.3. ÍNDICE BITMAP

Nessa estrutura de índice um vetor de bits é criado para cada valor único da coluna indexada. Cada bitmap contem um único bit (0 ou 1) para cada tupla da relação. O “1” indica que a linha tem o valor da coluna representado pelo bitmap, “0” indica o contrário.

O índice bitmap permite a rápida recuperação de registos cujas colunas indexadas têm baixa cardinalidade e que são freqüentemente consultadas juntas. Além disso operações com bitmap ocorrem no nível de bits sendo portanto mais eficientes.

No caso de uma transação não é possível nesse tipo de índice bloquear um único bit, e consequentemente os bloqueios são feitos a nível de bloco. Sabendo-se que um grande número de linhas (ou bits) podem estar em um único bloco, muitas linhas serão bloqueadas. Tal fato torna o índice bitmap não adequado para aplicações com um número moderado de transações.

2.3.ÍNDICES EM BANCOS DE DADOS DE DOCUMENTOS

A chave de um índice é composta por uma ou mais colunas de uma relação, no entanto o tamanho dessa chave tem um limite, pois quanto maior for essa chave maior será o espaço necessário para o seu armazenamento. Teoricamente, o tamanho máximo de uma chave de índice será o tamanho de um bloco, mas ao se implementar índices com chaves tão grandes deve-se considerar a questão do número de operações de entrada/saída que serão necessárias para percorrer o índice no momento da busca.

Em um banco de dados de documentos, cada documento é representado como sendo um valor de determinado campo em um registro, geralmente uma coluna do tipo BLOB. Como os documentos podem ter tamanho imprevisível, essas colunas não podem ser utilizadas como chave de um índice. Além disso na maioria das vezes o usuário precisará executar buscas por frases completas e/ou fragmentos de texto além da tradicional busca por palavra ou parte de uma palavra. Nestes casos as estruturas de índices implementadas como árvore B e índice bitmap são insuficientes para suportar esses níveis de consulta. Levando-se em consideração que cada documento pode ter milhares de palavras, e apenas uma relação de documentos pode conter milhares de documentos, torna-se um desafio a criação de um índice para permitir a busca nesses documentos.

Entre as técnicas utilizadas para indexar documentos destacam-se *arquivos invertidos com contadores de posição* e *índices para a próxima palavra*. Um arquivo invertido possui duas partes principais: a primeira delas corresponde a uma estrutura de busca, chamada de vocabulário, contendo todos os termos distintos presentes no texto indexado. A segunda parte corresponde a uma lista invertida, mantida para cada termo, que armazena a referência para os documentos contendo aquele termo. Essa Segunda parte também armazena, em cada

referência, as respectivas posições onde o termo ocorre no texto para permitir a busca por frase.

As buscas são executadas através da conjunção ou disjunção entre as listas relativas aos termos presentes na consulta.

Nos índices para a próxima palavra, para cada palavra existente no vocabulário é criada uma lista contendo as palavras que ocorrem na posição subsequente do texto, juntamente com apontadores de posição para essas ocorrências.

2.3.1.ÍNDICES INVERTIDOS COM MARCADORES DE POSIÇÃO

Segundo HARMAN et. al. (1992) arquivos invertidos são tradicionalmente utilizados para a implementação de índices lexicográficos. Para permitir a busca por frases o arquivo invertido contém, para cada palavra, um apontador para cada um dos documentos nos quais a palavra ocorre, juntamente com as posições da palavra nesse documento. Nesse tipo de índice a busca torna-se bem mais eficiente, mas ela traz consigo um custo adicional, a dimensão do próprio índice. Segundo HARMAN et. al. (1992) essa dimensão é da ordem de 10 a 100% do tamanho do texto original. Além disso, o índice precisa ser atualizado a cada modificação dos documentos originais. A Figura 4 mostra um exemplo de um arquivo invertido, a referência para os documentos e as posições onde os termos ocorrem no documento.

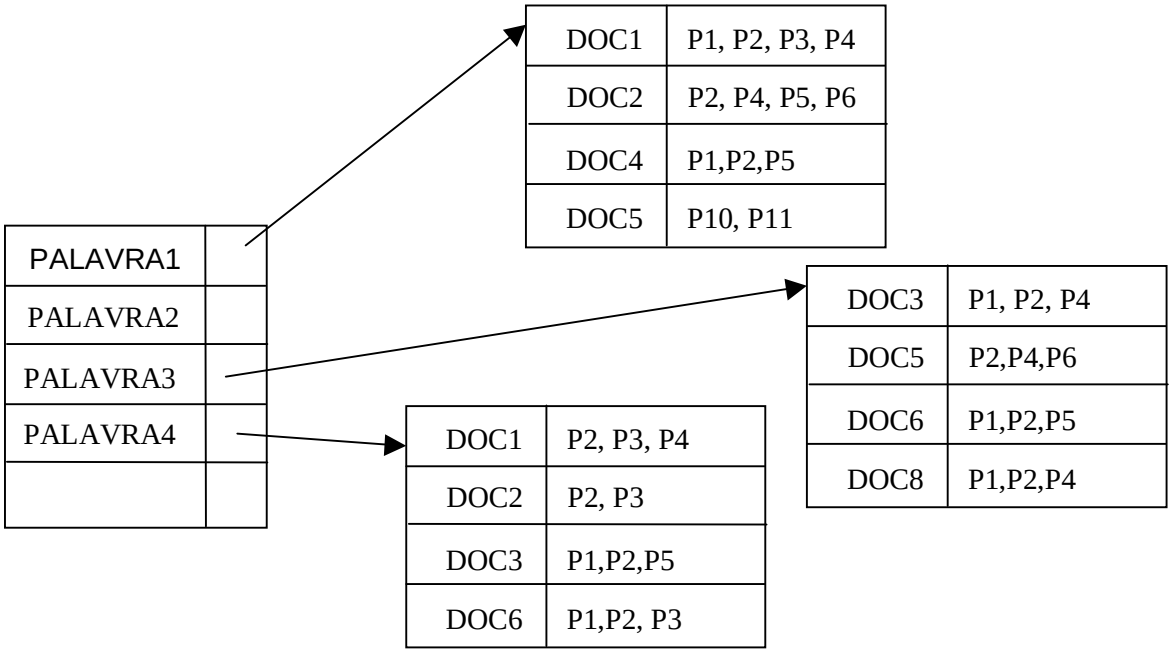


Figura 4 - Índice invertido

Nos índices invertidos as pesquisas por frases são feitas da seguinte forma:

(1) O primeiro termo é pesquisado, como resultado tem-se uma lista temporária de documentos e posições contendo o termo pesquisado.
(2) Utiliza-se a lista temporária para pesquisar o próximo termo, sendo retirados dela todos os documentos cujo o termo pesquisado não ocorre na posição adequada, ou seja, na posição subsequente ao termo anteriormente pesquisado.
(3) Repete-se a pesquisa para os próximos termos até que o último tenha sido pesquisado, ou até que a lista esteja vazia, indicando que a frase não foi encontrada.

Quanto menor for o número de documentos em que os primeiros termos pesquisados ocorrem, menores serão as listas temporárias e mais rápidas serão as pesquisas dos próximos termos. Dessa forma é desejável que os termos menos comuns sejam pesquisados inicialmente.

2.3.2.ÍNDICES PARA A PRÓXIMA PALAVRA

Segundo HUGH et. al. (1999) um índice para a próxima palavra consiste de um vocabulário de palavras distintas e, para cada uma dessas palavras *w*, uma lista com as próximas palavras que a sucedem. A lista de próximas palavras consiste de todas as palavras *s* que sucedem *w* em algum local da base de documentos. Para cada par *ws* existe uma seqüência de localidades, determinando o documento e a posição em que esse par ocorreu.

A Figura 5 mostra um exemplo de índice para a próxima palavra. Na primeira árvore tem-se a lista das palavras que ocorrem no banco de dados e que possuem uma palavra subseqüente. Para cada palavra da primeira árvore tem-se um ponteiro para uma segunda árvore denominada árvore de próximas palavras. Nessa segunda árvore cada nó tem um ponteiro para uma lista dos documentos onde o par de palavras ocorre. A lista de documentos tem o identificador do documento e as posições onde o par de palavras ocorre no documento e, é através dessas posições que são feitas buscas por frases.

Para pesquisar a frase *"para sempre vou lembrar"*, inicialmente deve-se determinar os pares a serem pesquisados. Agrupa-se as palavras aos pares para que seja efetuada a pesquisa no índice. Assim, realiza-se a recuperação do par *"para sempre"*, e depois do par *"vou lembrar"'*. Observando a Figura 5, nota-se que o par *"para sempre"'* ocorre no documentos 5 nas posições 2 e 23. E o par *"vou lembrar"* ocorre no mesmo documento na posição 4, ou seja, duas posições subseqüentes a 2. Isso significa que os pares se sucedem no texto. Se a

consulta desejada fosse a frase *"para sempre vou"*, não seria possível dividir a frase em pares distintos. Então se pesquisaria pelos pares *"para sempre"'* e *"sempre vou"*. Esses pares, para serem considerados sucessivos, devem ocorrer com a diferença de uma posição.

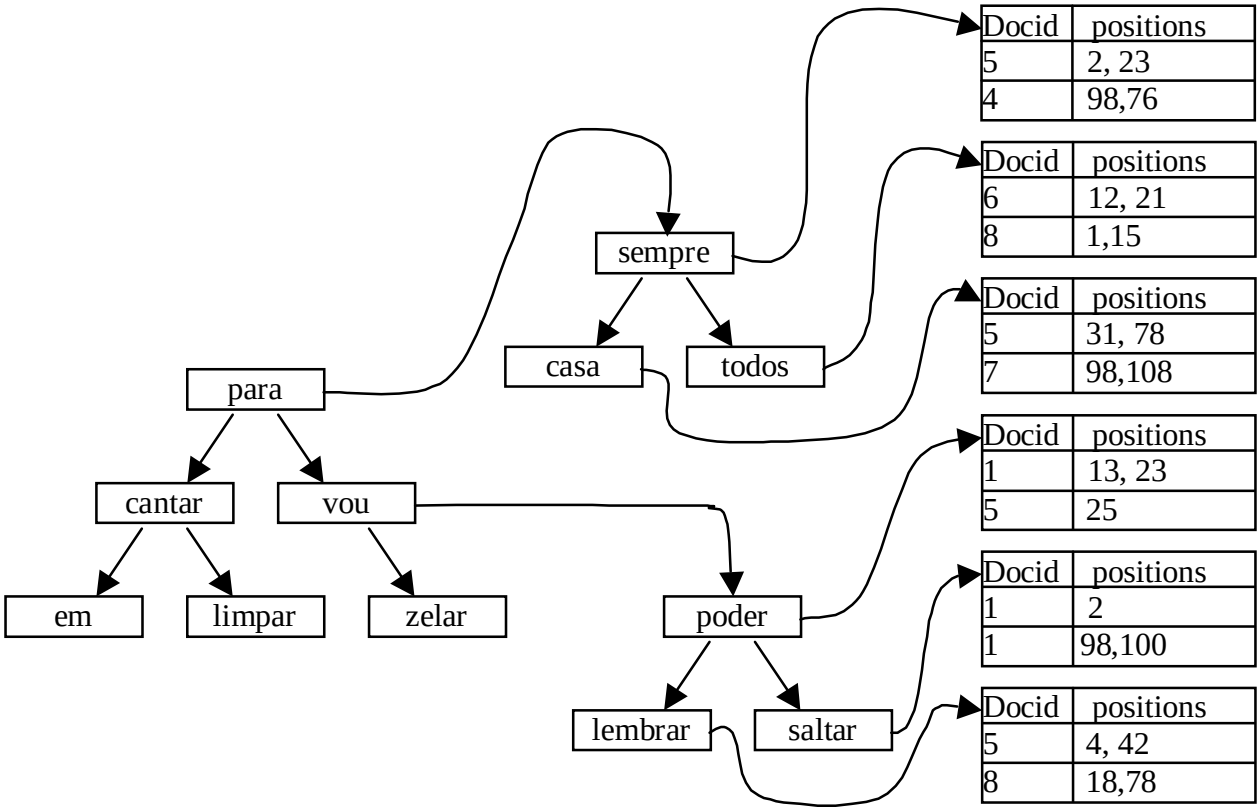


Figura 5 – Índice para a próxima palavra

3. FIREBIRD

3.1. HISTÓRIA

O Firebird surgiu em julho de 2000 quando a Borland disponibilizou o código fonte do Interbase, seu banco de dados comercial. Nessa época a norte-americana Ann W. Harrison que participou, desde os primeiros dias, do desenvolvimento do Interbase, juntou-se a Mark O'Donohoue da Austrália e Mike Nordell da Suécia e, criaram uma árvore CVS (Sistema de Controle de Versões) de código no site sourceforge.net para que pudessem trabalhar no processo de implementação e compilação. Desde então vários desenvolvedores têm contribuído para a evolução e aperfeiçoamento do Firebird.

3.2. ARQUITETURA INTERNA

As distribuições do Firebird se dividem em duas arquiteturas: Classic e SuperServer.

A arquitetura *Classic*, como próprio nome já sugere "Arquitetura clássica", foi a primeira implementada pela equipe de desenvolvimento do Firebird e foi desenvolvida para ser utilizada em sistemas operacionais que não dispunham de tecnologia para executar aplicações multitarefas. Nessa arquitetura quando um cliente tenta se conectar ao Firebird uma instância do executável do banco é iniciada e fica dedicada ao cliente, enquanto a conexão estiver ativa. O gerenciamento de bloqueio é feito através de um programa separado, chamado gds_lock_mgr, que é iniciado quando o segundo cliente se conecta ao mesmo banco. Esse programa garante o bloqueio dos recursos do banco para os clientes.

O servidor cria uma instância do executável para cada cliente conectado. Assim cada cliente mantém seu próprio cache das páginas do banco por ele utilizadas. Para se comunicar com os clientes o servidor, na arquitetura clássica, mantém uma área de memória compartilhada.

A arquitetura SuperServer é baseada na tecnologia de aplicações multitarefas onde apenas uma instância do executável do banco é iniciada. Nessa arquitetura uma thread é responsável por esperar uma requisição de conexão, enquanto uma outra tem a função de gerenciar o bloqueio dos recursos do banco. Um único espaço de cache é utilizado por todos os clientes otimizando o uso da memória.

Nesse SGBDR os dados são armazenados em páginas, as quais podem ser de diversos tipos, tais como, página de índice, log, dados e blob, dentre outros. Por convenção a página 0 do arquivo principal é sempre a de cabeçalho do banco. Cada página tem seu próprio cabeçalho que informa, entre outras coisas, o seu tipo e data/hora da última atualização.

Os dados de um banco no Firebird podem estar fisicamente armazenados em um ou mais arquivos, sendo que um deles é o arquivo principal, que contém, entre outras, informações sobre a definição do banco.

Quando o Firebird se conecta a um banco de dados, inicialmente ele lê os primeiros 1024 bytes do arquivo principal. Isso ocorre porque a página de cabeçalho armazena informações críticas sobre o banco, tais como tamanho da página e número de versão do ODS (On Disk Structure), nos primeiros 1024 bytes. Após se certificar de que o arquivo é um banco de dados Firebird válido e que o número de versão do ODS é compatível com o servidor corrente, ele lê a página de cabeçalho utilizando o tamanho correto da página e recupera outras informações.

As páginas de dados armazenam registros ou fragmento de registros. Cada registro ou fragmento tem seu próprio descritor que é armazenado de cima para baixo na página, enquanto que os registros são preenchidos do final da página para o início. A página está cheia quando esses dados se encontram.

Cada página de dados só pode conter registros de uma única tabela, embora o descritor de cada registro informe a relação a qual ele pertence.

Além da descrição, cada registro possui também um cabeçalho que armazena o identificador da transação que escreveu o registro e um flag para informar entre outras coisas se o registro foi excluído ou se está corrompido. A descrição também armazena a versão do formato dos metadados, que define o número, tipo de dados e ordem de cada campo na tabela.

A tabela RDB$FORMATS armazena um histórico de todos os formatos para cada tabela. Assim, quando o formato de uma tabela é modificado uma nova versão é criada e os registros daquela relação só serão atualizados na próxima vez que forem acessados.

A alocação de páginas no Firebird é feita através da rotina PAG_allocate cuja declaração aparece no arquivo jrd/pag.cpp na versão em C++ do Firebird ou jrd/pag.c na versão escrita em C. Essa função procura pela primeira página com espaço disponível e se for necessário ela lê uma nova página do disco e cria uma imagem para a mesma no gerenciador de cache, retornando um ponteiro para a página reservada na memória.

3.3.ÍNDICES NO FIREBIRD

O Firebird suporta índices bitmap e de navegação. O primeiro deles é implementado utilizando-se vetores de bits, enquanto que o outro é implementado utilizando-se árvores B+.

Todos os índices, independentemente do tipo dos mesmos, são armazenados em páginas do banco que são classificadas como *index root page* e *index pages*. As páginas chamadas de *index root page* contêm informações sobre cada índice de uma tabela, além de armazenar um ponteiro para a página raiz do índice e a seletividade do índice. As informações sobre a descrição da chave do índice são mantidas nas tabelas RDB$INDICES e RDB$INDEX_SEGMENTS.

Index pages são as páginas que armazenam o índice efetivamente, são elas:

(1) A página raiz do índice que tem ponteiros para as páginas do próximo nível do índice.

(2) As páginas dos níveis intermediários que armazenam além das chaves e do número da página corrente, um ponteiro para a próxima página no mesmo nível. Esse ponteiro é utilizado para se percorrer o índice da esquerda para a direita sem que seja necessário consultar as páginas do nível acima.

(3) As páginas do último nível do índice, que representam as folhas, armazenam além do valor das chaves informações para localização de cada registro.

3.4.MANIPULAÇÃO DE BLOB NO FIREBIRD

O Firebird permite a criação de BLOBs como dados associados a um campo de uma tabela. Daí a existência de um tipo de dado com esta denominação. Como um BLOB pode ter seu tamanho indeterminado, o que realmente é armazenado como valor do campo é a identificação do mesmo, ou seja o blobid. Os dados são armazenados separadamente em páginas especiais em outro local do banco.

Um blobid pode ser temporário ou permanente. Temporário é aquele que foi criado, mas ainda não foi associado como valor de campo para a linha de uma tabela. Já os permanentes são aqueles que já foram associados como valor de um campo em um registro de alguma tabela.

O blobid é composto por 8 bytes que permite ao Firebird identificar e, consequentemente, localizar os dados do BLOB. Os primeiros 4 bytes representam a identificação da tabela a qual o ele está relacionado. Os últimos 4 bytes representam a

identificação dentro da tabela. Para qualquer BLOB temporário a identificação do relacionamento é 0.

Um tipo de página especial armazena os dados reais do BLOB. No caso dos dados excederem essa página, ela não conterá dados e sim ponteiros para outras páginas de dados.

Para cada BLOB criado é definido um registro que contém a localização dos dados e algumas informações sobre o conteúdo do mesmo, que serão úteis no momento da recuperação dos dados.

O mecanismo de armazenamento é determinado pelo número do nível (0, 1 ou 2) do BLOB, que está associado ao tamanho do mesmo. Todo BLOB é criado como sendo do nível 0, mas pode se transformar no nível 1 ou 2 conforme aumente seu tamanho.

O de nível 0 é aquele que pode ser completamente armazenado na mesma página do registro de cabeçalho do mesmo. Uma página de dados de 4096 bytes representaria um BLOB de aproximadamente 4052 bytes.

Quando um BLOB é muito grande para caber em uma única página, então ele será de nível 1 e a página inicial do registro irá conter um vetor de números de páginas.

O nível 2 é utilizando quando a página inicial do registro não é suficiente para armazenar o vetor de todas as páginas. Nesse caso, o Firebird irá criar múltiplas páginas de vetores que podem ser acessadas através da página inicial. O tamanho máximo de um BLOB de nível 2 é o produto entre o máximo número de páginas de ponteiros, o número das páginas de dados por página de ponteiros e o espaço disponível em cada página de dados.

A manipulação de dados em um BLOB pode ser feito utilizando-se ferramentas de programação como Delphi ou C++Builder, que dispõem de componentes especiais especificamente desenvolvidos para acessar o banco de dados. Além disso na internet existem componentes gratuitos que oferecem acesso ao Firebird. Ainda assim, pode-se criar funções embutidas em DLLs (Dynamic Link Library) que façam a manipulação de dados em campos dessa natureza. Além desses recursos muitas outras linguagens podem ser utilizadas para acessar o Firebird através da utilização de *drives* de comunicação.

4. TECNOLOGIA UTILIZADA NO MECANISMO DE BUSCA GOOGLE

Segundo Brin and Page (1998, p. 1) a quantidade de informações disponíveis na Web está crescendo rapidamente e ao mesmo tempo o número de usuários inexperientes na arte da busca, por informações específicas, na Web também. Daí a necessidade de se dispor um mecanismo de busca que permitisse recuperar informações precisas sobre determinado assunto e ao mesmo tempo sem comprometimento da performance. Assim nasceu o Google, cujo nome vem da pronúncia em inglês da palavra googol ou 10^{100}. Ele opera sobre um gigantesco cluster Linux, contendo mais de 80 TB de capacidade de armazenamento em disco.

O Google tem duas principais características que ajudam a produzir resultados mais precisos nas buscas. Primeiro, ele utiliza a própria estrutura de links da Web para quantificar a importância de cada página em relação as outras. Essa classificação é chamada de PageRank. Como segunda característica tem-se que ele utiliza os links da Web para melhorar o resultado da pesquisa.

Segundo Page (1998, p. 1) a importância de uma página Web é uma coisa inerentemente subjetiva, que depende do interesse, conhecimento e propósito do leitor. Mas há ainda muito que pode ser dito objetivamente sobre a importância relativa das paginas Web.

Segundo Haveliwala (1999, p. 01) a idéia essencial por trás do PageRank é que se a pagina *u* tem um link para a página *v*, quer dizer que o autor de *u* está implicitamente atribuindo alguma importância à página *v*. Sendo *Nu* o número de links que saem de *u* e *PageRank(p)* a importância da página *p*. Então o *link (u, v)* confere *PageRank(u)/Nu* unidades de classificação a *v*.

Normalmente em um mecanismo de busca o texto de uma âncora (link para outra página) é armazenado como sendo parte da página onde a âncora está. No Google esse mesmo texto é associado a página que o link aponta. A razão para esse fato, segundo Brin and Page (1998), é que as âncoras freqüentemente trazem mais informações sobre as páginas referenciadas do que elas mesmas e podem existir âncoras para documentos que não podem ser indexados por um mecanismo de busca, tais como imagens, programas e bancos de dados.

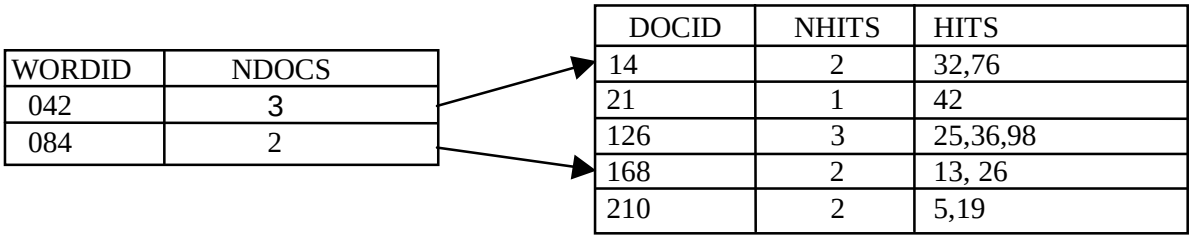
Segundo Brin and Page (1998) as estruturas de dados do Google foram projetadas para que uma grande quantidade de documentos pudessem ser baixados da internet, indexados e pesquisados com baixo custo.

O Google mantém na memória principal um vocabulário com mais de 14 milhões de palavras. Esse vocabulário é composto por duas partes: a primeira delas é uma lista de

palavras concatenadas e separadas por nulo e a outra, uma tabela hash de ponteiros. Cada ponteiro está associado a uma palavra, identificada por um número inteiro denominado wordid, e aponta para um índice invertido, contendo a lista de documentos com a respectiva lista de ocorrências da palavra dentro de cada documento. Essa tabela é também utilizada para converter cada palavra em um wordid.

Cada documento é identificado por um número inteiro chamado docid. O checksum da URL de cada documento é mantido em um arquivo junto com seu respectivo docid. Dessa forma um docid pode ser localizado basicamente executando-se uma busca binária no arquivo contendo o par checksum-docid.

O índice invertido é criado utilizando-se o wordid e o docid, como pode ser visto na Figura 6. Para cada ocorrência da palavra identificada por wordid, em um documento, é gerada uma lista de ocorrências, onde cada ocorrência é codificada em dois bytes armazenando, entre outras informações, a posição no documento onde a palavra ocorre.



WORDID	NDOCS
042	3
084	2

DOCID	NHITS	HITS
14	2	32,76
21	1	42
126	3	25,36,98
168	2	13, 26
210	2	5,19

Figura 6 - Índice invertido

Para executar uma busca, cada termo da consulta é convertido em um wordid que é utilizado como entrada para pesquisa na tabela hash e tendo como resultado um ponteiro para a lista de documentos contendo o termo identificado por wordid. O resultado final da busca é dado pela conjunção ou disjunção entre as listas de documentos das palavras pesquisadas, observando-se os critérios da consulta.

5. DESENVOLVIMENTO DO ÍNDICE

5.1. REQUISITOS PRINCIPAIS

O índice proposto deve comportar as informações necessárias à busca por frases e por proximidade de palavras, de forma que toda a lógica da busca pode ser feita no índice sem que seja necessário consultar os arquivos de dados. Os únicos dados acessados são aqueles que venham a atender aos critérios da busca. Além disso, a estrutura deve permitir a modificação do índice sem comprometer a performance das buscas.

Como o índice será projetado para um banco de dados ele deve permitir acessos simultâneos.

5.2. ESPECIFICAÇÃO DO PROTÓTIPO

O índice proposto é composto por um vetor de 16 milhões de entradas, onde cada saída do vetor aponta para um índice invertido. Esse vetor é mantido na memória principal em um espaço de 64 MB.

As palavras indexadas não precisam ter tamanho limitado, ao invés disso, as palavras são quebradas em termos de tamanho fixo. Assim cada termo indexado deve ter no máximo 4 caracteres de tamanho, as palavras maiores serão quebradas em partes para que sejam armazenadas no índice. Cada caracter é compactado em 6 bits, tendo-se assim 64 representações possíveis para cada caracter, entre letras e números. Assim, são necessários apenas 24 bits para representar cada termo o que dá 16 milhões de termos possíveis, que é exatamente o tamanho do vetor. Além disso, um inteiro de 4 bytes é suficiente para armazenar cada termo, e tal inteiro é utilizado como índice do vetor para o termo.

A compactação utilizada é necessária para que o vetor possa ser armazenado em 64MB, mas o custo disso é que não é possível representar todos os caracteres possíveis com apenas 6 bits, por isso o índice aqui proposto só armazena letras e números totalizando 36 representações possíveis. Entretanto com 6 bits temos 64 representações possíveis, ou seja, as 28 representações possíveis que sobram podem e devem ser utilizadas. Essas representações poderiam ser utilizadas para representar caracteres de pontuação por exemplo.

Cada entrada do vetor representa um termo e cada saída aponta para uma lista de documentos onde aquele termo ocorre, se a saída do vetor for nula quer dizer que aquele termo não está presente em nenhum documento do banco. A lista de documentos contém além

do identificador, as posições e o número de vezes que o termo ocorre em cada documento. As informações sobre as ocorrências dos termos são armazenadas em um vetor de inteiros de 256 elementos, de 0 a 255, onde o primeiro elemento guarda o total de ocorrências do termo no documento.

O índice invertido é implementado utilizando-se árvore B+, para tornar a busca mais eficiente.

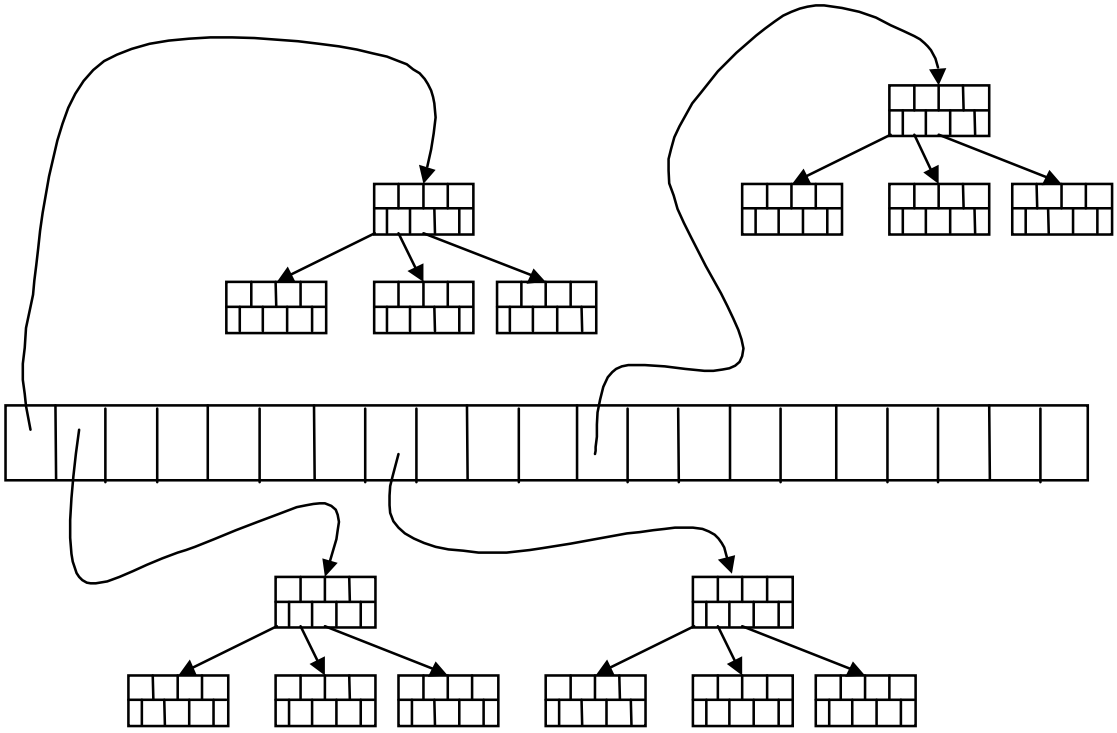


Figura 7 - Índice proposto

Sendo x um termo de no máximo 4 caracteres, $f(x)$ a função que retorna o inteiro representado por x, e v o vetor que tem como saída ponteiros para listas de documentos. A lista l_i de documentos onde o termo x_i ocorre é dada pela função $l_i = v[f(x_i)]$. Se l_i for vazia não existem documentos onde x_i ocorre, caso contrário os documentos estarão na lista l_i.

Supondo que se deseje pesquisar pela palavra “ajuda”, sendo esta maior que quatro caracter divide-se a mesma em dois termos “ajud” e “a”, após essa divisão converte-se os termos em números inteiros, obtêm-se as listas de documentos contendo os termos procurados usando-se os inteiros como índice no vetor, faz-se então a interseção das listas, observando-se as posições dos termos procurados. Nesse caso o temo “a” deve ser encontrado quatro posições subsequentes ao termo “ajud” no mesmo documento.

A exclusão de um documento no índice pode ser feita a qualquer momento, sem que seja necessário suspender as buscas no momento da exclusão. Isso é possível através da exclusão lógica do documento, que é feita, simplesmente, zerando-se as ocorrências de palavras para o documento no índice.

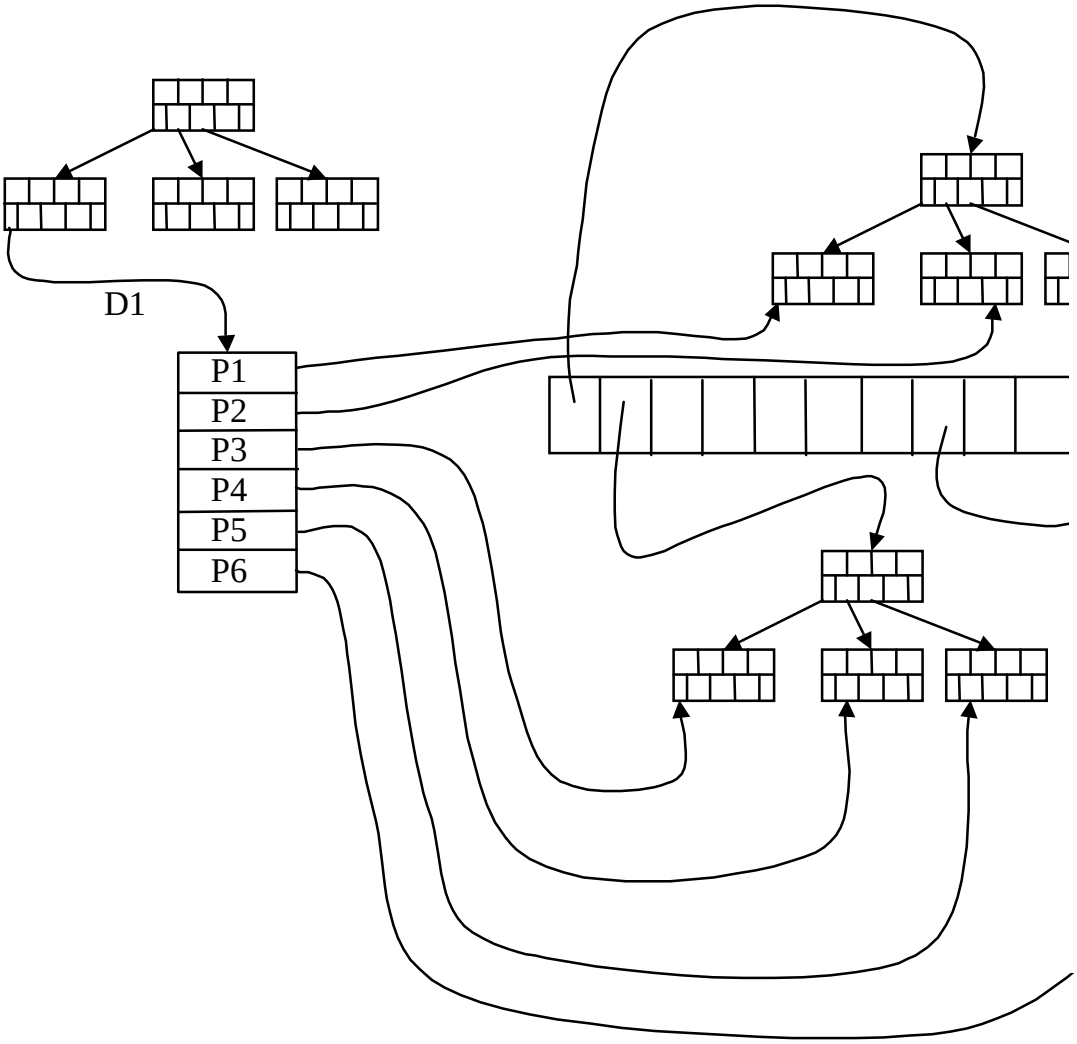


Figura 8 - Estrutura auxiliar utilizada para atualizar o índice.

O identificador dos documentos indexados é mantido em uma árvore B+ e cada folha tem uma lista de ponteiros para as ocorrências do documento no índice. Assim quando um documento é excluído ou alterado, essa lista é utilizada para atualizar as ocorrências do documento no índice sem que o mesmo seja acessado diretamente. A desvantagem da exclusão lógica é que o espaço marcado como excluído não será mais utilizado, criando-se

espaços vazios na árvore, até que ocorra uma reindexação e as árvores sejam recriadas. A Figura 8 apresenta uma ilustração da estrutura auxiliar utilizada na exclusão lógica.

Existe ainda uma lista de palavras comuns que não precisam ser indexadas porque aparecem na maioria dos documentos e por isso só ocuparia mais espaço no índice. Essa lista de palavras é mantida em um arquivo separadamente e no momento da indexação ela é utilizada para evitar que as palavras nela presentes e contidas no documento sejam indexadas.

5.3. ADEQUAÇÃO AO FIREBIRD

Para que a estrutura de índice proposta seja implementada no Firebird os seguintes critérios devem ser observados:

(1) O índice deverá ser armazenado nas páginas do SGBD, como é feito com quaisquer outro tipo de índice no Firebird. Assim, o tamanho dos nós que são implementados como árvore B+ devem ser ajustados para que caibam em uma página do banco, e isso é possível simplesmente ajustando-se a quantidade de chaves possíveis de serem armazenadas em cada nó. Quanto ao vetor de 64 MB, deve ser investigada uma forma de representá-lo em páginas também, possivelmente utilizando-se algum tipo de compressão para reduzir o seu tamanho. Inicialmente sugere-se que sejam armazenados apenas as palavras que possuem um ponteiro saindo do vetor. E para cada palavra deve ser armazenado o número de página do índice invertido que guarda as ocorrências da mesma.

(2) Deverá ser acrescentado ao código fonte do Firebird nos arquivo idx.c e idx.h as funções para manipular o índice proposto. Em tais arquivos são encontradas atualmente as funções necessárias para manipular os índices B-Tree. Nos arquivos blb.c e blb.h, deverão ser acrescentadas algumas funções para acessar os blobs de forma conveniente, mesmo sabendo-se que já existem muitas funções disponíveis em tais arquivos que são destinadas à manipulação de páginas especiais de BLOB. Algumas funções do arquivo btr.c, o qual define o gerenciamento de árvores B+, deverão ser utilizadas para gerenciar os nós das árvores B+ que armazenam os índices invertidos.

(3) As funções delete_blob_index(blobid), update_blob_index(blobid) e update_all_blob_index(struct bloblist *root_bloblist) deverão ser adaptadas e incluídas no código para excluir, atualizar e recriar um índice. A atualização do índice deve ser permitida de duas maneiras. Nessa versão do protótipo a função update_all_blob_index(struct bloblist *root_bloblist) que recria o índice pode ser adaptada e anexada ao código do SGBD para permitir a recriação do índice. A função

update_blob_index(blobid) deverá ser utilizada para permitir a reindexação de um BLOB quando o mesmo for atualizado. Finalmente a função add_blob(char *filename, struct bloblist *blob_info) deverá ser ajustada para receber um identificador de BLOB. Tal função na implementação corrente do índice é utilizada para indexar um arquivo, e deve ser adaptada passando a indexar um BLOB.

(4) O código do arquivo exe.c que define a execução dos comandos SQL deve ser modificado, acrescentando-se o código necessário à exclusão, atualização e criação do índice.

(5) Deverá ser acrescentado ainda o código necessário ao reconhecimento, por parte da linguagem SQL do SGBD, da sintaxe a ser definida que será utilizada para pesquisar frases.

6. EXPERIMENTO

6.1. DESCRIÇÃO DO EXPERIMENTO

O experimento consistiu em testar a indexação de documentos e a busca por palavras e frases nos documentos indexados.

Para executar os testes utilizou-se um PC com processador AMD de 700 MHz, 128 MB de memória RAM e disco rígido de com 4 GB de espaço total e 1,21 GB de espaço livre.

Foi utilizado um programa, gratuito e de livre distribuição, que gera palavras aleatórias para criação dos arquivos a serem indexados. Esse programa é chamado Insult Writer 2000 encontra-se disponível no site <ftp://ftp.externet.hu/pub/mirror/sac/utiltext/insult2k.zip>.

6.2. TESTES

Para fins ilustrativos o Gráfico 1 mostra experimentalmente o tempo consumido na indexação de alguns arquivos.

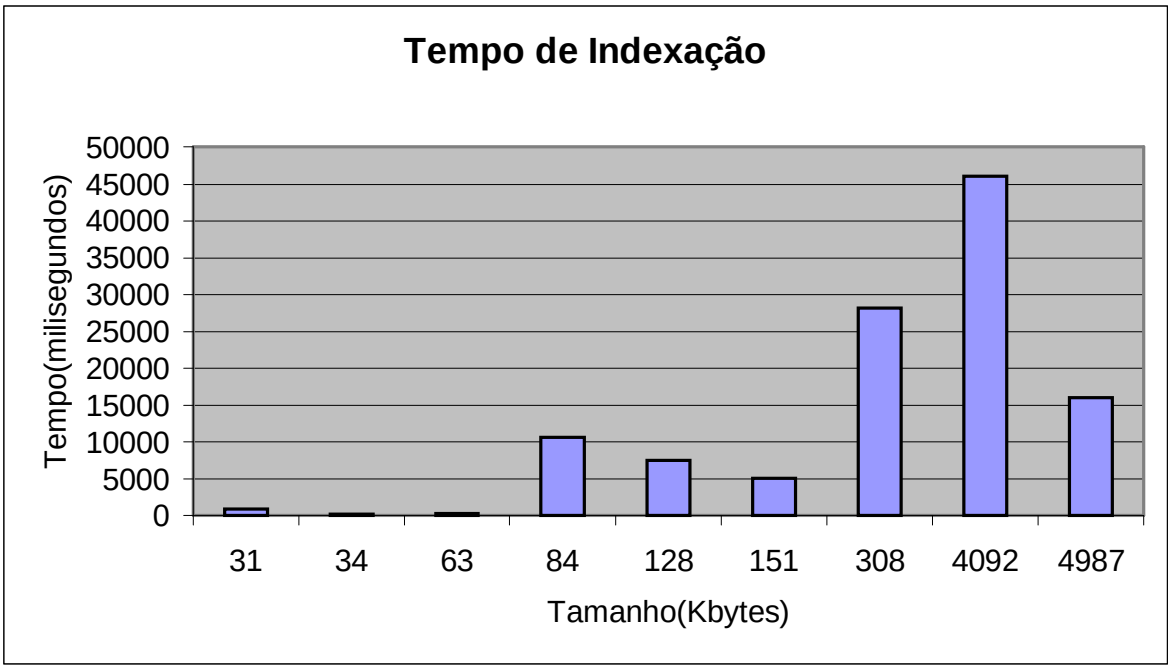


Gráfico 1 – Tempo de Indexação de documentos.

No gráfico os arquivos aparecem na mesma ordem em que foram indexados, da esquerda para a direita. Observando o gráfico é possível notar que o arquivo de 34 KB levou menos tempo para ser indexado que o arquivo de 31 KB, o mesmo ocorreu com o arquivo de 4987 KB. Isso ocorre porque quando uma palavra é indexada pela primeira vez, uma árvore é

criada para armazenar as ocorrências da mesma nos documentos, e isso leva tempo. Mas nas próximas vezes que a mesma palavra for inserida no índice apenas alguns campos serão atualizados, somente quando o nó a ser inserido estiver cheio é que ocorrerá uma divisão do mesmo. Entretanto vias de regra para registrar a ocorrência de uma palavra já presente no índice levará menos tempo que para indexar uma palavra não presente no índice.

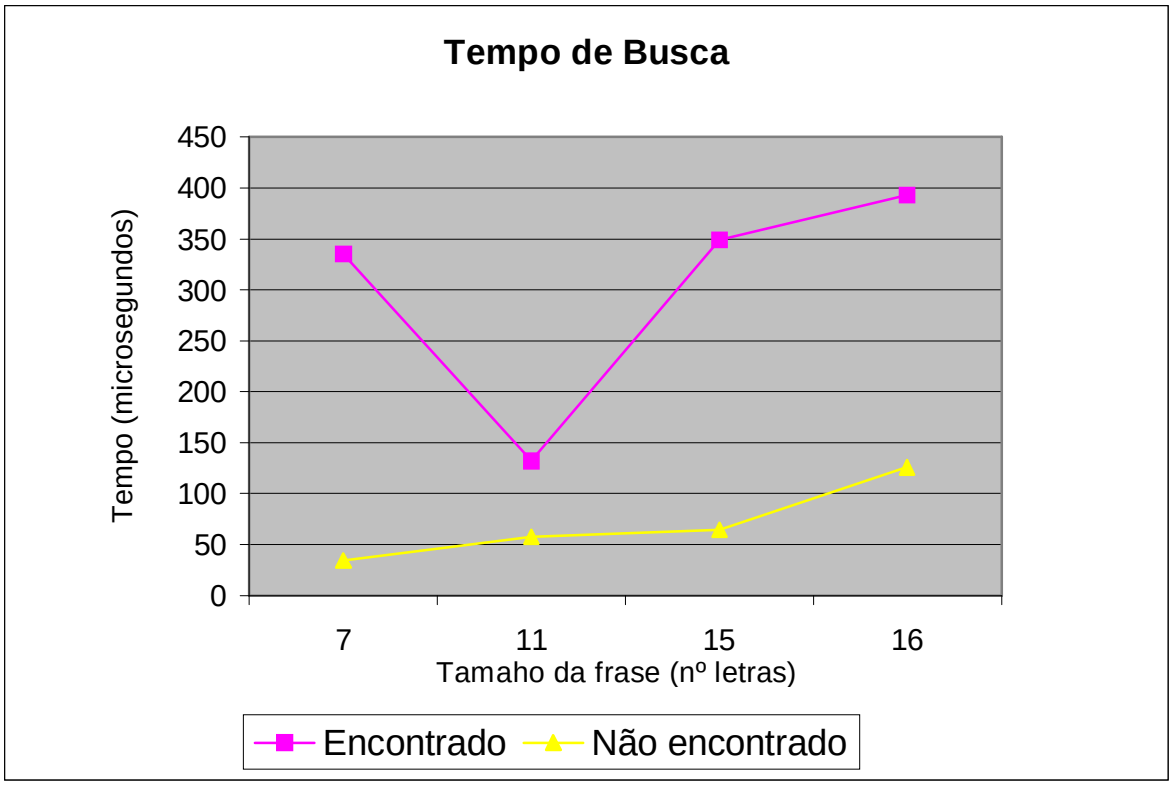


Gráfico 2 – Tempo de busca de documentos.

O Gráfico 2 mostra o tempo consumido na busca. Esse tempo corresponde às operações de interseção das listas de documentos onde os termos pesquisados ocorrem, observando-se as posições dos mesmos. Observa-se no gráfico que as frases não encontradas levam menos tempo para serem pesquisadas. Isso se deve, em parte, ao fato de que a maior parte do tempo consumido na busca corresponde ao tempo gasto com comparação de listas de documentos onde os termos procurados ocorrem, Como a lista de resultados tende a ficar vazia antes do final da busca, os tempos diminuem.

6.3.AVALIAÇÃO

Os dados mostrados nos gráficos 1 e 2 são meramente ilustrativos, no entanto é possível notar que à medida que o índice cresce em diversidade de palavras mais rápido fica o processo de indexação de novos documentos.

Por outro lado, conforme previsto, os testes mostraram que o índice ocupou mais espaço do que o tamanho original dos arquivos indexados.

A corrente versão do protótipo utiliza apenas a memória principal para armazenar o índice. Assim, quando a memória não é suficiente o SO faz paginação, o que reduz a performance do índice. Porém a estrutura em árvore B+ que é utilizada para armazenar o índice invertido permitirá, em um trabalho futuro, que esses nós sejam armazenados como páginas do SGBD, eliminando a necessidade de paginação por parte do SO.

Quando o protótipo implementado é executado ele carrega um vetor de inteiros que ocupa cerca de 64 MB da memória principal. Depois disso ele preenche todas as entradas do vetor com zeros, esse preenchimento é feito com uma função especial presente na linguagem C e que é específica para preencher uma região da memória com um valor. Depois disso o vetor está pronto para o uso. O vetor é preenchido com endereços de memória a medida que os termos vão sendo indexados. Quando um elemento do vetor tem seu valor diferente de zero quer dizer que esse valor representa o endereço, na forma de um número inteiro, da árvore alocada na memória.

A ordem em que os arquivos são indexados influencia os tempos gastos para indexar os mesmos. Isso ocorre porque quando um termo é indexado pela primeira vez, um nó para a árvore de ocorrências do mesmo é alocado, porém tal nó tem a capacidade de armazenar 10 ocorrências do mesmo termo, assim nas próximas nove ocorrências desse termo o mesmo nó será utilizado, e o tempo gasto com a alocação será evitado. Uma nova alocação só será feita quando o nó estiver cheio ocorrendo então a criação de um novo nó com a distribuição das ocorrências entre os mesmos.

7. CONCLUSÕES

O conhecimento técnico e científico obtidos, fruto da interdisciplinaridade do tema proposto, por si só, independentemente dos resultados obtidos, já justificam a relevância desse trabalho.

Ele compreende três perspectivas distintas: A primeira corresponde ao entendimento das diversas teorias que possibilitaram a união de conceitos distintos de modo a se conceber uma estrutura que fosse capaz de atender aos requisitos de busca e performance aceitáveis. A outra está relacionada a implementação do modelo proposto utilizando os conhecimentos mais aprofundados sobre estruturas de dados e programação. Finalmente a perspectiva dos testes e validação do modelo que possibilitou o refinamento do mesmo.

A complexidade dos assuntos envolvidos e o exíguo tempo para o desenvolvimento do trabalho foram fatores restritivos à concretização dos objetivos originalmente propostos.

Nos periódicos, revistas e livros pesquisados não foi encontrada nenhuma abordagem sobre a implementação de índices invertidos utilizando-se árvores B+ e função hash.

Constatou-se nesse trabalho a viabilidade da implementação de um índice sobre arquivos de texto longo sem a necessidade de recursos de hardware mais sofisticados, visto que o protótipo apresentado foi implementado em um PC (Computador Pessoal) convencional.

A estrutura de dados proposta e implementada mostrou-se capaz de suportar buscas por frases em documentos de texto longo.

A linguagem C, utilizada na implementação do protótipo, mostrou-se bastante adequada, pois dispunha dos recursos imprescindíveis à implementação, como aritmética de ponteiros e acesso a arquivos. Além disso C é a linguagem na qual o Firebird foi originalmente desenvolvido.

A implementação do protótipo não levou em consideração aspectos de performance e compressão de dados, por estarem além dos objetivos desse trabalho. Porém, tais aspectos constituem temas relevantes para serem desenvolvidos em trabalhos posteriores.

Embora muitos estudos estejam sendo realizados na área de indexação de documentos, o estudo da indexação de campos do tipo BLOB não foi encontrada.

Quanto ao algoritmo de busca por frase utilizado, escolheu-se o índice invertido por se tratar de uma estrutura que permite a indexação incremental de documentos sem a necessidade de bloquear o acesso ao índice. Tal estrutura mostrou-se adequada à implementação do protótipo por oferecer menor custo de manutenção nas tarefas de

atualização do índice. Além disso o uso de árvores B+ na implementação dos índices invertidos torna-o adequado ao armazenamento em páginas do banco.

A utilização de um vetor de endereçamento direto, para localizar o índice invertido de cada termo, tornou o processo de busca mais eficiente. Visto que a etapa de pesquisa no índice pelo termo procurado é eliminada, o que ocorre na prática é apenas a interseção das listas de documentos que contém os termos procurados.

Quanto aos resultados obtidos nos testes pode-se concluir que entre as variáveis que podem influenciar o tempo de indexação de um arquivo, destacam-se o tamanho das palavras e a quantidade de termos repetidos no mesmo. Em relação ao tempo de busca, esse está diretamente associado ao tamanho das listas a serem comparadas, assim sugere-se que os primeiros termos pesquisados sejam os mais raros possíveis, tornando as listas menores e reduzindo o tempo gasto com comparações.

7.1. DIFICULDADES ENCONTRADAS

No percurso do desenvolvimento do índice foram encontradas muitas dificuldades das quais pode-se destacar:

a) Em relação a compilação do Firebird 1.0.2, mesmo seguindo o manual de compilação que acompanha o código fonte, ocorreram vários erros relativos a características próprias do Sistema Operacional (SO) e em outros casos o próprio código fonte do Firebird precisou ser modificado corrigindo-se erros de implementação.
b) Documentação sobre o Firebird incompleta, faltando manuais avançados sobre o funcionamento interno do mesmo.
c) No início do trabalho o código fonte do Firebird estava na versão 1.0.2, a qual foi escrita em C, era muito complexa, mal documentada e tinha vários recursos desativados mas que ainda não tinham sido fisicamente retirados do código, tornando-o mais extenso e reduzindo, consequentemente, a legibilidade do mesmo. Tal código já havia sido condenado pela própria equipe de desenvolvimento do Firebird que atualmente concentra esforços na conclusão da nova versão 1.5, conforme pôde ser observado no site http://firebird.sourceforge.net acessado em 26/08/2003. Com lançamento previsto para meados de agosto/2003, essa nova versão está sendo rescrita utilizando-se a linguagem C++ com técnicas de orientação a objetos. Como a versão 1.0.2 era a mais recente disponível até a conclusão desse trabalho, tornou-se inviável a implementação do protótipo no código fonte do Firebird, conforme originalmente se pretendia.

7.2.LIMITAÇÕES

O protótipo apresentado não leva em consideração questões de performance como a quantidade de operações de entrada/saída ou ocupação da memória em função do crescimento do índice. Além disso essa versão do protótipo não garante a redução no tamanho do índice em relação ao tamanho do arquivo indexado, já que nenhuma técnica de compactação é utilizada, exceto na representação dos termos.

A implementação atual do índice não suporta pesquisa por partes de uma palavra, porém sugere-se o desenvolvimento desse tema em trabalhos futuros. Tal busca poderia ser feita mantendo-se uma estrutura auxiliar com a lista de palavras indexadas. No momento da busca a lista seria consultada retornando o conjunto de palavras onde a substring aparece, nesse ponto as palavras seriam pesquisadas no índice.

7.3.SUGESTÕES

Sugere-se como trabalhos futuros a pesquisa e desenvolvimento dos seguintes itens:

a) Avaliação da performance do índice;
b) O estudo sobre a possível utilização de alguma técnica de compactação aplicada às árvores B+, para otimizar o uso dos recursos.
c) Adequar o índice para que o mesmo possa ser armazenado em páginas do banco.
d) Modificar o código fonte do Firebird acrescentado ao mesmo o índice aqui proposto.
e) Avaliar o comportamento do índice implementado no Firebird, no que diz respeito a performance das consultas, alterações no índice e concorrência
f) Acrescentar ao índice a capacidade de indexar caracteres de pontuação
g) Modificar a linguagem SQL, para suportar pesquisas por frases.

8. APÊNDICE A – CÓDIGO FONTE DO PROTÓTIPO

No presente apêndice são apresentadas algumas partes do código fonte do protótipo implementado.

O Quadro 1 mostra a função hash utilizada para converter uma palavra em um inteiro de quanto bytes.

Quadro 1 - Função Key_Hash usada na conversão de uma palavra em wordid.

```
/*************************************************************
          key_hash(char *word)
       Função que pega a palavra word
    e retorna uma chave a ser utilizada na tabela hash

 *************************************************************/

unsigned long int key_hash(char *word){
      short aux;
      register int i;
      unsigned long int wordid;
      aux ='a';
      wordid = 0;
   for(i=0;(char)word[i]!=(char)NULL;i++){
            wordid = (wordid << 6) + ((word[i] - aux)<0?(word[i] - '0'):(word[i] - aux));
       }
       return wordid;
}
```

O Quadro 2 mostra a função utilizada para localizar a árvore referenciada pelo wordid de uma palavra no vetor de acesso direto.

Quadro 2 – Função lookup_hah usada para acessar o a árvore B+.

```
/*************************************************************
          lookup_hash(unsigned long int wordid)
       Função que pega uma palavra identificada por wordid
    e retorna um ponteiro para uma árvore B+ contendo a
       lista de documentos onde a palavra indicada por wordid
       é encontrada.

 *************************************************************/
struct BTree * lookup_hash(unsigned long int wordid){
      return (struct BTree *)*(hash_table + wordid);
}
```

Quadro 3 - Função utilizada para inserir uma chave na árvore B+.

```
/*************************************************************
insert_key(struct BTree *root, struct BTree *p, struct RootFather *fatherlist,
                        int key,int *rec, int pos)
 Função que insere uma chave na árvore B+ apontada por root  e retorna um
ponteiro para a nova árvore. Essa função divide  os nós se for necessário.
 *************************************************************/
struct BTree *insert_key(struct BTree *root,struct BTree *p, struct RootFather
*fatherlist,int key,int *rec, int pos){
        struct BTree *nd, *nd2, *f;
        struct BTree *newnode = NULL;
        typehits newrec;
        int midkey, newkey;
        typehits midrec;
        copyhits(newrec, rec);
        newkey = key;
        nd = p;
        f = getfather(fatherlist);
        /* enquanto não chegou a raiz (f=NULL) e o nó corrente
   estiver completo divida-o */
        while(f!=NULL && nd->numnodes == max_nodes){
          nd2=getnode();
          split(nd, pos, newkey, newrec, newnode, nd2, &midkey, midrec);
          if(!nd->nodes[0])
                nd->nodes[nd->numnodes - 1] =nd2;
          newnode = nd2;
          pos = index(midkey, f);
          nd = f;
          f = getfather(fatherlist);
          newkey = midkey;
          copyhits(newrec, midrec);
          /*newrec = midrec;*/
        }
        /* encontrou um nó incompleto insere um nó filho */
        if(nd->numnodes < max_nodes){
                insnode(nd, pos, newkey, newrec, newnode);
           return root;
        }
        /* f é NULL e nd->numtrees é max_nodes de modo que
   nd é uma raiz completa, nesse caso divide a raiz e cria uma nova raiz */
   nd2=getnode();
        split(nd, pos, newkey, newrec, newnode, nd2, &midkey, midrec);
        nd->nodes[nd->numnodes - 1] =nd2;/* ligação entre as folhas */
   root = maketree(midkey,midrec);
   root->numnodes = 2;
   root->nodes[0] = nd;
   root->nodes[1]= nd2;
        return root;
}
```

O Quadro 4 mostra a função usada para pesquisar uma frase no índice.

Quadro 4 – Função utilizada para pesquisar por uma palavra ou frase.

```
/*************************************************************
      void search_word_list(char *search_list){
      Pesquisa por uma palavra ou frase no .
 *************************************************************/
void search_word_list(char *search_list){
  char *list;
  char *word;
  int position;
  struct doc_list *list_found;
  list = malloc(strlen(search_list)+1);
  strcpy(list,strupr(search_list));
  list_found = NULL;
  word=strtok(list," ");
  position=0;
  if(word!=NULL){
      do {
        list_found = search_token( word, list_found);
        word=strtok(NULL," ");
      }while (word!=NULL && list_found!=NULL);
  }
  show_results(list_found);
  free(list);
}
```

9. REFERÊNCIAS BIBLIOGRÁFICAS

BAHLE, D.; WILLIAMS H. E.; ZOBEL, J. **Optimised Phrase Querying and Browsing of Large Text Database.** Proceedings of the Australasian Computer Science Conference, M. Oudshoorn (ed), Gold Coast, Australia, Janeiro de 2001.

BEACH, Paul. **InterBase and Blobs - A Technical Overview.** IBPhoenix. Disponível em: <http://www.ibphoenix.com/main.nfs?a=ibphoenix&page=ibp_blobs>. Acesso em: 16/07/2003.

BEACH, Paul. **Structure of a Data Page**. IBPhoenix. Disponível em: <http://www.ibphoenix.com/main.nfs?a=ibphoenix&page=ibp_ods_page>. Acesso em: 15/08/2003.

BRIN, Sergey et al. **The PageRank Citation Ranking: Bringing Order to the Web.** 1998.
NAGARAJARAO, A.; GANESH, J. R.; SAXENA, A. **An Inverted Index Implementation Supporting Efficient Querying and Incremental Indexing.** 6 de Maio de 2002.

BRIN, Sergey; PAGE, Lawrence **The Anatomy of a Large-scale Hypertextual Web Search Engine.** In Proceedings of the Seventh International World Wide Web Conference, 1998.

CHAN, H.; YASHKIR, D. **Conceptual Architecture for InterBase/Firebird**. Faculty of Mathematical, Statistical and Computer Sciences. University of Waterloo, Canada. 29 de janeiro de 2002. Disponível em: <http://www.ibphoenix.com/main.nfs?a=ibphoenix&page=ibp_waterloo>. Acesso em: 15/07/2003.

DATE, C. J. **Introdução a Sistemas de Banco de Dados.** Rio de Janeiro. Campus, 1991.
Wiederhold, G. **Database Design.** New York. McGraw-Hill, 1982.

ELMASRI, Ramez.; NAVATHE, Shamkant B. **Fundamentals of Database Systems**. Addison-Wesley, 1999.

HARMAN, D. Et al. **Inverted Files, in Information Retrieval - Data Structures & Algorithms.** Prentice Hall, 1992.

HARRISON, Ann. **High-level Description of the InterBase 6.0 Source Code.** IBPhoenix. Disponível em: <http://www.ibphoenix.com/a549.htm>. Acesso em: 20/06/2003.

HARRISON, Ann. **Space Management in InterBase.** Disponível em: <http://www.ibphoenix.com/main.nfs?a=ibphoenix&page=ibp_spam>. Acesso em: 06/08/2003.

HARRISON, Ann; BEACH, Paul. **A Cut Out and Keep Guide to the Firebird Source Code**. IBPhoenix. Disponível em: <http://www.ibphoenix.com/main.nfs?a=ibphoenix&page=ibp_source_guide>. Acesso em: 05/07/2003.

HAVELIWALA, Taher H. **Efficient Computation of PageRank.** Stanford University. Outubro 18, 1999.

HUGH E. W.; ZOBEL J.; ANDERSON P**. What's Next? Index Structures for Efficient phrases quieryng.** Proceedings of the Tenth Australasian Database Conference, Auckland, New Zeland, Janeiro 18-21, 1999.

NEGUS, Ernie. Insult Writer 2000. Disponível em: <ftp://ftp.externet.hu/pub/mirror/sac/utiltext/insult2k.zip>. Acesso em: 10/08/2003.

PUTZ , Steve. **Using a Relational Database for an Inverted Text Index.** XEROX,1991.

SANTOS, Paulo V. **Características e Especificações do SGBD Firebird 1.0**. Comunidade Firebird de Língua Portuguesa. Disponível em: <http://www.comunidade-firebird.org/cflp/downloads/CFLP_O014.PDF>. Acesso em: 16/08/2003.

TENENBAUM, A.M.; LANGSAM, Y.; AUGENSTEIN, M.J. **Estruturas de Dados Usando C.** São Paulo. Makron Books, 1995.