UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

COORDENAÇÃO DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

CHRISTIANE SMOKOVICZ

LEANDRO MALAQUIAS SPRENGER DE BARROS

MOTORES DE BUSCAS EM REDES SOCIAIS: UM ESTUDO COMPARATIVO DAS TECNOLOGIAS SQL LIKE, MYSQL FULL-TEXT SEARCH E LUCENE

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2011

CHRISTIANE SMOKOVICZ

LEANDRO MALAQUIAS SPRENGER DE BARROS

MOTORES DE BUSCAS EM REDES SOCIAIS: UM ESTUDO COMPARATIVO DAS TECNOLOGIAS SQL LIKE, MYSQL FULL-TEXT SEARCH E LUCENE

Trabalho de Conclusão de Curso apresentada como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas da Coordenação de Análise e Desenvolvimento de Sistemas da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Ademir Mazer Jr.

PONTA GROSSA

2011

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Ponta Grossa

Diretoria de Graduação e Educação Profissional

TERMO DE APROVAÇÃO

MOTORES DE BUSCAS EM REDES SOCIAIS: UM ESTUDO COMPARATIVO DAS TECNOLOGIAS SQL LIKE, MYSQL FULL-TEXT SEARCH E LUCENE.

por

CHRISTIANE SMOKOVICZ

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 09 de novembro de 2011 como requisito parcial para a obtenção do título de Tecnóloga em Análise e Desenvolvimento de Sistemas. A candidata foi arguida pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Ademir Mazer Junior
Prof.(a) Orientador(a)

Gleifer Vaz Alves
Membro titular

Helyane Bronoski Borges
Membro titular

Helyane Bronoski Borges
Responsável pelos Trabalhos
de Conclusão de Curso

André Koscianski
Coordenador(a) do Curso
UTFPR - Campus Ponta Grossa

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Ponta Grossa

Diretoria de Graduação e Educação Profissional

TERMO DE APROVAÇÃO

MOTORES DE BUSCAS EM REDES SOCIAIS: UM ESTUDO COMPARATIVO DAS TECNOLOGIAS SQL LIKE, MYSQL FULL-TEXT SEARCH E LUCENE.

por

LEANDRO MALAQUIAS SPRENGER DE BARROS

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 09 de novembro de 2011 como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Ademir Mazer Junior
Prof.(a) Orientador(a)

_____
Gleifer Vaz Alves
Membro titular

_____
Helyane Bronoski Borges
Membro titular

_____
Helyane Bronoski Borges
Responsável pelos Trabalhos
de Conclusão de Curso

_____
André Koscianski
Coordenador(a) do Curso
UTFPR - Campus Ponta Grossa

AGRADECIMENTOS

Agradecemos a Deus, por sua infinita sabedoria e confiança em nós depositadas e pela oportunidade de estarmos nesta vida evoluindo a cada dia e superando nossas dificuldades. Amém.

Á nossas famílias que são nossa fonte de carinho, compreensão, apoio e amor.

Aos nossos amigos que tanto os estimamos e pelos momentos de descontração, trabalhos que realizamos e conhecimentos compartilhados.

Ao nosso orientador por acreditar e apoiar o desenvolvimento deste estudo.

Á coordenação do curso e todos os professores que contribuíram significativamente em nossas carreiras acadêmicas.

E um ao outro, por acreditar que este trabalho foi possível em meio a nosso namoro e demonstrar que além de namorados, somos amigos e companheiros de ideias.

RESUMO

SMOKOVICZ, C. BARROS, L. M. S. de. **Motores de Buscas em Redes Sociais: um estudo comparativo das tecnologias SQL LIKE, MYSQL Full-Text Search e LUCENE**. 2011. 79 p. Trabalho de Conclusão de Curso Superior em Análise e Desenvolvimento de Sistemas - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2011.

Neste trabalho são apresentados os motores de busca disponíveis atualmente na *Web* sendo: o operador *LIKE* da linguagem SQL que é utilizada na maioria dos SGBDs, os índices *full-text* implementados no SBGD *MySQL* e a biblioteca de indexação e pesquisa *Lucene*; Devido à utilização crescente das redes sociais pelos usuários da Web e o volume de informações nas redes sociais crescendo exponencialmente, a organização e a disponibilização destas informações para posterior recuperação de maneira ágil, precisa e relevante são pontos importantes a serem observados e estudados. Para os testes de desempenho com estes motores de busca, foi considerada uma rede social onde o objeto social é um cenário habitado por indivíduos que estudam e/ou trabalham distantes de suas cidades natais e buscam opções de moradia, entretenimento e serviços emergenciais. Os resultados a partir dos testes de desempenho apresentaram a biblioteca de indexação e pesquisa *Lucene* como melhor solução a ser implementada na rede social proposta neste trabalho, pois os tempos mínimo e máximo para atender uma requisição de busca são melhores aos demais motores de busca.

Palavras-chave: Busca. *Lucene*. Índices. *MySQL full-text search*. *SQL LIKE*.

ABSTRACT

SMOKOVICZ, C. BARROS, L. M. S. de. **Search Engines in Social Networks: a comparative study of technologies LIKE SQL, MYSQL Full-Text Search and Lucene**. 2011. 79 p. Trabalho de Conclusão de Curso Superior em Análise e Desenvolvimento de Sistemas - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2011.

In this work are presents the search engines that are currently available on the Web: the LIKE operator of SQL language that is used in most DBMSs, the full-text indexes implemented in SGBD MySQL and the Lucene library of indexing and searching; Due to the increased use of social networks by Web users and the volume of information in social networks growing exponentially, the organization and the availability of this information for later retrieval in a fast, accurate and relevant way are important points to be observed and studied. For the performance tests with these search engines, was considered a social network where the scenario is inhabited by a people who study and / or working away from their hometowns and seek housing options, entertainment and emergency services. The results from performance tests showed the Lucene library of indexing and searching as the best solution to be implemented in the social network proposed in this paper, because the minimum and maximum times to serve a search request are better than others search engines.

Keywords: Search. *Lucene*. Indexes. *MySQL full-text search*. *SQL LIKE*.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API - *Application Programming Interfaces* (Interface de Programação de Aplicativos)

IDE - *Integrated Development Environment* (Ambiente de Desenvolvimento Integrado)

FTP - *File Transfer Protocol* (Protocolo de Transferência de Arquivos)

HTTP - *HyperText Transfer Protocol* (Protocolo de Transferência de Hipertexto)

HTML – *HyperText Markup Languag*e (Linguagem de Marcação de Hipertexto)

IP – *Internet Protocol* (Protocolo de Internet)

JSON - *JavaScript Object Notation* (Notação de Objeto JavaScript)

JSP - *Java Server Pages* (Páginas de Servidor Java)

JSTL - *Java Server Pages Standard Tag Library* (Biblioteca Padrão de Tags de Páginas de Servidor Java)

JVM - *Java Virtual Machine* (Máquina Virtual Java)

RAM – *Random Access Memory* (Memória de acesso aleatório)

RDBMS - *Relational Database Management System* (Sistema Gerenciador de Banco de Dados Relacional)

SGBD - Sistema Gerenciador de Banco de Dados

SQL – *Structured Query Language* (Linguagem de Consulta Estruturada)

TF-IDF – *Term Frequency - Inverse Document Frequency* (Freqüência do Termo-Freqüência Inversa do Documento)

URL – *Uniform Resource Locator (*Localizador Padrão de Recursos*)*

XML - *Extensible Markup Language* (Linguagem de Marcação Extensível)

SUMÁRIO

1 INTRODUÇÃO

O ser humano necessita ampliar suas relações através da comunicação entre as pessoas, buscar e compartilhar informações entre os indivíduos ao meio em que ele vive. As redes sociais oferecem oportunidade para as pessoas possuírem um espaço próprio e personalizado; necessidade de expansão de contato e assim os usuários convidam seus amigos para participar na rede social (BOLETINES PANDALABS, 2008).

Rompendo barreiras geográficas e temporais, as novas formas de comunicação e de participação dos usuários têm crescido exponencialmente, formando as chamadas “comunidades virtuais”, que possibilitam a seus usuários a compartilharem informações e interagirem entre si. E também, a encontrarem seu “nicho” social, onde possam expressar sua visão de vida e exercer sua originalidade, participando do contexto social via suas manifestações personalizadas.

Este trabalho explora o “nicho” dos indivíduos que estudam e/ou trabalham em cidades distantes da sua cidade natal e propõe uma ferramenta como solução para a centralização destas informações sendo parte da solução do problema enfrentado por eles. Porém, para esta centralização de informações ocorrer de maneira eficiente, é preciso levantar todos os requisitos que esta ferramenta necessita.

Para a elaboração deste trabalho, verificou-se que não há trabalhos relacionados e buscou-se na literatura, os conceitos e o embasamento teórico necessário para a contextualização do objeto social proposto que visa sua implementação na Internet, sendo um ambiente de interação e/ou um espaço público complementar. Sendo que, sua complexidade e abrangência, com vínculos que não se delimitam as fronteiras geográficas e culturais (etnias, religião, idioma, gênero, etc.), oferece novas formas relacionais e diferentes possibilidades de apropriação das tecnologias de informação e comunicação.

A partir da pesquisa e da elaboração da contextualização, ao elencar todos os requisitos necessários da rede social a ser desenvolvida, verificou-se a criticidade da busca, apresentando dificuldades na organização e posterior recuperação das informações visando a agilidade, precisão e relevância. Percebe-se então, que a pesquisa contextualizada de informações de uma rede social é um ponto crítico, pois engloba todas as informações e apresenta-as seguindo critérios fornecidos pelo usuário.

Este trabalho apresenta três motores de busca (*SQL*, *SQL full-text* e *Lucene*), que são prototipados e verificados, através de testes com uma ferramenta que avalia o desempenho, qual a melhor solução para a organização destas informações na rede social proposta. São feitas conclusões acerca de todo o trabalho e apresentadas propostas de trabalhos futuros.

2 CONTEXTUALIZAÇÃO

Nas seções seguintes são apresentados e explicados os conceitos de *Web* 2.0 e de redes sociais, contendo um pequeno histórico. Na seção 2.3 sobre as redes sociais digitais e o fenômeno das comunidades virtuais na seção 2.3.1. Na seção 2.4 a busca da informação e o crescente volume de informações na *Web*, e uma forma de organização de informação, a indexação na seção 2.5.

2.1 *WEB* 2.0

"*WEB* 2.0" é o termo mais difundido dentro da indústria da tecnologia como sinônimo de sites colaborativos. A editora e a empresa de comunicação *O'Reilly Media* lançou o termo *Web* 2.0 como nome de uma conferência nos Estados Unidos no ano de 2004. O termo, que faz um trocadilho com o tipo de notação em informática que indica a versão de um software, foi popularizado pela *O'Reilly Media* e pela *MediaLive International* como denominação de uma série de conferências que tiveram início em outubro de 2004 (O'REILLY, 2005).

O termo se refere a uma mudança na forma de como a *Web* é encarada por usuários e desenvolvedores, englobando diversas linguagens em um ambiente de colaboração, interação e participação. Baseada em inteligência coletiva, isto é, construção coletiva do conhecimento. Por meio da interação, comunidades criadas em torno de interesses específicos poderão apoiar uma causa, discutir temas individuais ou de relevância coletiva, disseminar informações culturais, entre muitas outras ações (LUVIZOTTO e VIDOTTI, 2010).

A *Web* 2.0 facilitou em grande escala a criação de todo tipo de conteúdo e isso proporcionam ao usuário atuar como leitor, autor e produtor de conteúdo informacional e inclusive editor e colaborador. Nesse contexto, os arquivos, documentos, informações organizadas em banco de dados ficam disponíveis *online*, e podem ser acessados em qualquer lugar e momento, ou seja, não existe a necessidade de gravar em um determinado computador os registros de uma produção ou alteração na estrutura de um texto. As alterações são realizadas automaticamente na própria *Web* (BLATTMANN e SILVA, 2007).

A *Web 2.0*, como uma evolução na maneira de utilizar a *Web*, tem causado enorme impacto sobre a organização da informação em ambientes digitais, sobretudo, porque se criou uma grande expectativa de seus membros seriam eles

próprios capazes de coordenar os fluxos informacionais e o aprovisionamento de soluções no âmbito da gestão dos sistemas de informação (MOURA, 2009).

2.2 REDES SOCIAIS

As redes sociais são uma forma complexa de relacionamento entre indivíduos, grupos ou organizações agrupados em torno de valores, crenças ou interesses comuns. O desenvolvimento das redes sociais ocorre a partir da interação e comunicação entre os participantes da rede, o que a configura como uma construção social (TOMAÉL, 2007). Para Marteleto (2001, p.72), redes sociais são “um conjunto de participantes autônomos, unindo ideias e recursos em torno de valores e interesses compartilhados”.

Desde o início do século XVI, ponto de partida para o escritor Burke (2003) para seu livro sobre história social do conhecimento, já existia a preocupação dos intelectuais da época e membros de ordens religiosas (pesquisadores, médicos, advogados, entre outros), acerca da aquisição, construção e transmissão da informação. A partir de mosteiros e instalações de universidades, se formavam grupos de estudiosos que no decorrer dos próximos séculos, vão se consolidando com os secretários, bibliotecários, arquivistas, conselheiros que tinham função de acessar e organizar o conhecimento.

As redes de informação também se formavam nos portos quando os viajantes traziam dados geográficos de outras terras, continentes, mares, culturas, línguas disseminados em mapas e guias (SPUDEIT, 2009). Burke (2003) versa sobre a trajetória da construção do conhecimento e redes de informação na sociedade englobando importantes revoluções intelectuais, a exemplo do Renascimento, da Revolução Científica e do Iluminismo da Europa moderna, que foram movimentos culturais importantes que colaboraram com o fortalecimento das redes de informação e distribuição do conhecimento ao longo dos séculos. Antes, a rede era um fenômeno localizado, hoje se torna a base de uma compreensão da sociedade contemporânea. As redes estão translaçadas na sociedade, nas relações sociais, tecnológicas e virtuais.

De modo geral, entende-se uma rede social como uma estrutura social composta de atores, representados por pessoas, organizações ou territórios que se mantém conectados por um ou vários tipos de relações baseadas em amizade,

família, relações comerciais, entre muitas outras e, por meio dessas ligações vão construindo e re-construindo a estrutura social, partilham crenças, conhecimento ou prestígio (MOLINA e AGUILAR, 2004).

2.3 REDES SOCIAIS DIGITAIS

Em meados de 1990, o relacionamento entre os indivíduos passou também a ocorrer digitalmente, através de ferramentas tecnológicas disponíveis na Internet, onde as pessoas podem trocar informações, compartilhar experiências, colaborar com projetos, participar no aprendizado coletivo, fortalecer os laços entre seus membros e aumentar o poder de decisão do grupo. Surgem assim, as redes sociais digitais, que são ambientes dinâmicos, com participação na produção e veiculação de informação, de incentivo a participação e assim como em ambientes não virtuais tais redes também podem ter momentos de conflitos e lutas de interesse (ROCHA, 2005).

As primeiras redes sociais digitais surgiram com o *SixDegrees.com* em 1997, um site que permitia a criação de perfil virtual combinado com registro e publicação de contatos. Este site foi pioneiro, pois combinou funcionalidades que outros sites ofereciam, e foi descontinuado 3 anos depois, por problemas financeiros (BOYD e ELLISON, 2007). De 1997 a 2001, várias ferramentas de comunidade que combinavam perfis e listas públicas de amigos foram lançadas. Uma delas foi o *LiveJournal*, lançado em 1999, que permite que os usuários criem "jornais pessoais", com notícias de seu interesse. Outra foi a *Cyworld*, fundado em 1999 e que implementou funcionalidades de redes sociais em 2001 e ainda o sueco *LunarStorm*, que se tornou um site de rede social em 2000 (BOYD e ELLISON, 2007).

A próxima geração de sites de rede social veio com o *Ryze.com*, o *Fotolog* e o *Friendster*. *Ryze.com*, criado em 2001, ajudava as pessoas a montar redes sociais de negócios. O *Fotolog* surgiu em 2002, com uma idéia semelhante à de um blog. O *Friendster*, criado em 2002, focava em amigos de amigos (*friends-of-friends*), enquanto todos os sites sociais da época focavam em encontros amorosos entre estranhos. Com seu sucesso, em apenas um ano o *Friendster* já possuía mais de 30 milhões de usuários (BOYD e ELLISON, 2007). Contudo, tamanho sucesso acabou acarretando dificuldades técnicas, o que abriu espaço para novos serviços do

gênero, lançados entre 2003 e 2005, como *Facebook*, *Twitter*, *Orkut*, entre outros (XAVIER e CARVALHO, 2011).

Atualmente, as redes sociais em ambientes digitais são cada vez mais amplas, complexas e estruturadas. Toda rede possui uma temática que serve de motivação e aglutinação de seus participantes, e que se desdobra em subtemas gerados por interesses específicos que vão surgindo ao longo do seu desenvolvimento (MARTÍNEZ, 2000).

Uma rede social virtual é normalmente um ponto de encontro de usuários que tem interesses em comum. Estas redes permitem a gravação de perfis, com o usuário podendo disponibilizar informações sobre si (textos, fotos, imagens, vídeos, etc.). A conexão entre os usuários é realizada através destes perfis, que podem ser visualizados por outros (na verdade, por quem também tiver outro perfil). Para facilitar a comunicação dos usuários existem os grupos, que podem definir afinidades, abrindo espaço para discussões e debates, apresentando temas variados (AZEVEDO, 2009). Com esta interação entre os indivíduos, na rede social digital nota-se o fenômeno de comunidades virtuais.

2.3.1 Comunidades Virtuais

As comunidades virtuais são um conjunto de pessoas que estão interligadas entre si de forma democrática a partir de um propósito previamente definido e que utilizam o ciberespaço como um dos ambientes para a troca de experiências, informações e construção de novos saberes. As comunidades virtuais são constituídas, principalmente, pelas relações de colaboração e cooperação entre os membros dos grupos que a compõem. São as relações de colaboração e cooperação que mantêm as comunidades virtuais vivas. Enquanto existirem tais relações, as comunidades sobreviverão podendo, inclusive gerar novas comunidades (TAJRA, 2002).

De acordo com Barros (1994), as atitudes colaborativas são interações ocorridas entre duas ou mais pessoas que contribuem de alguma forma para a obtenção de objetivos que não, necessariamente, sejam de interesse comum. Segundo Maturana (1997), as interações são consideradas o alicerce para a estruturação, composição e manutenção das comunidades virtuais.

A interação dos usuários realizada a partir das redes sociais é caracterizada não apenas pelas mensagens trocadas (o conteúdo) e pelos integrantes da rede que

se encontram em contextos geográfico, social, político e temporal diferentes. Esta interação é caracterizada também pelo relacionamento que existe entre os integrantes. Trata-se de uma construção coletiva, inventada pelos indivíduos que agem durante o processo, que não pode ser manipulada unilateralmente nem pré-determinada (PRIMO, 2007).

Lévy (2002) tem defendido a participação em comunidades virtuais como um estímulo à formação de inteligências coletivas, às quais os indivíduos podem recorrer para trocar informações e conhecimentos. Fundamentalmente, ele percebe o papel da comunidade quando convenientemente organizada, representa uma importante riqueza em termos de conhecimento distribuído, de capacidade de ação e de potência cooperativa. Nessas redes sociais, o valor da informação para uso ou reuso pode ser majorado não só através do registro de debates e opiniões sobre diversos assuntos, mas também pelo registro do próprio relacionamento entre os participantes e aproveitando as experiências anteriores de uso dos indivíduos relacionados a um usuário.

Os graus de participação dos usuários dependem: do interesse dos integrantes na temática da rede social, nos conteúdos nela veiculados e das ações comunicativas que propiciam a interação dos nós na rede. Mesmo nas redes orientadas por objetivos pré-definidos institucionalmente, não há possibilidade de previsão nem garantia de controle de todas as interações que nela vão surgir (AGUIAR, 2007).

Convém destacar, que a participação das pessoas em uma rede social, blog, ou *software* social é desigual, sendo 90% dos usuários ativos de leitores assíduos, mas não participam com conteúdo ou opinião; 9% dos usuários editam, comentam, participam, mas de forma ocasional, enquanto 1% publica e participa efetivamente (NIELSEN, 2006).

2.4 BUSCA DA INFORMAÇÃO

Ao longo da história da humanidade sempre se buscou encontrar formas de facilitar a execução das tarefas do cotidiano, para tal, elaboram-se mentalmente várias formas de organização e classificação de itens e objetos, atribuindo-lhes nomes, números, cores, etc., facilitando assim uma posterior identificação e recuperação destes. A classificação é uma atividade indispensável para se obter a

organização em um determinado ambiente. Classificar exige tempo e dedicação, pois é um processo que reúne objetos, ideias e/ou seres em grupos segundo suas diferenças e semelhanças (ALBUQUERQUE, MESQUITA e COSTA, 2010).

Devido ao seu caráter dinâmico, parece inviável tratar a informação na *Web* com os modelos de tratamento da informação tradicionais. Isso por uma série de fatores, como, por exemplo, o aumento do volume de dados que cresce exponencialmente chegando a ser impossível precisar a quantidade de informação veiculada neste ambiente. A magnitude de informações na *Web* tomou proporções tão gigantescas que tanto o gerenciamento como o tratamento de informação na Internet só pode ser feito com o auxílio de ferramentas e *softwares* especializados. O ato de representar um documento através de um conceito é de mesma natureza nos diferentes contextos – físicos e digitais, e a indexação é uma dessas formas de representar um documento (GUEDES e DIAS, 2010).

2.5 INDEXAÇÃO

De modo geral, a indexação é definida como um conjunto de procedimentos com objetivo de expressar/representar o conteúdo temático de documentos através de linguagens de indexação ou documentárias visando à recuperação posterior (LANCASTER, 2004). O produto da representação temática é denominado termos de indexação. Segundo Lancaster (2004), os termos atribuídos ao documento no processo de indexação servem como ponto de acesso mediante os quais um item é localizado e recuperado.

As linguagens de indexação são instrumentos convencionais de uso das unidades de informação para a descrição dos conteúdos temáticos dos documentos (GUINCHAT e MENOU, 1994), podendo ser classificadas como linguagens controladas; linguagens naturais e linguagens livres (ROWLEY, 2002).

- Linguagens controladas de indexação – definidas como um conjunto de termos autorizados para uso na indexação do assunto de documentos. É considerada a mais sistemática e eficiente proporcionando mais qualidade e facilidade de utilização, é usada em sistemas de recuperação da informação onde o processamento técnico ocorre manualmente e, portanto, com um custo elevado.

- Linguagens naturais de indexação – referem-se a quaisquer expressões que ocorram em alguma parte do documento. Todos os termos no corpo do documento são candidatos a serem termos de indexação. É a mais utilizada em sistemas de recuperação da informação automatizados onde os documentos se apresentam em formato digital, pode ser caracterizado pelo seu baixo custo e exaustividade presente nas buscas.
- Linguagens livres de indexação – para esta linguagem não existem limitações quanto aos termos a serem utilizados no processo de indexação. É usada por um indexador humano e sua qualidade irá depender do conhecimento do indexador sobre o(s) assunto(s) tratado no documento.

Para Hjorland (2001), a indexação deve ser moldada para se ajustar às necessidades de determinada clientela, indo além da escolha de linguagem de indexação adequada. A indexação também poder ser analisada tomando como perspectiva o agente executor do processo. Segundo Rafferty e Hidderley (2007) existem três grupos de candidatos a atores no processo de indexação, são eles (GUEDES e DIAS, 2010):

- Indexação orientada por especialistas – baseia-se no tratamento da informação através da intervenção de intermediários (bibliotecários, indexadores, editores voluntários), é a indexação feita por especialistas sendo dispendiosa e cara.
- Indexação orientada pelo autor – esta abordagem pressupõe que o autor irá utilizar termos que são comumente compreendidos e geralmente aceitos. Um problema que essa abordagem enfrenta é o fato do autor não ser necessariamente um gestor de informação com os conhecimentos profissionais de um especialista.
- Indexação orientada pelo usuário – esse tipo de indexação possibilita um elevado nível de interação com a comunidade que, provavelmente, não seria possível se tivesse decisões a serem feitas sobre códigos, convenções e regras que regem qualquer taxonomia controlada.

3 O OBJETO SOCIAL PROPOSTO

Nas seções seguintes é apresentado a motivação, o ciclo de vida e os requisitos do desenvolvimento deste trabalho, que propõem uma rede social como solução ao problema enfrentado por certos indivíduos.

3.1 MOTIVAÇÃO

Atualmente, há dezenas de sites que oferecem “serviços de redes sociais” cada qual buscando um “nicho de mercado” relacionado a algum tipo de subcultura (adolescentes, músicos, participantes de jogos baseados em avatares, entre outros), 15 dos quais concentram a audiência em todo o mundo, geoestrategicamente distribuídos (LE MONDE, 2008).

A principal motivação para a criação dessa ferramenta é explorar o “nicho” de indivíduos que são oriundos de outras cidades e que mudam de cidade seja para fins de estudo ou trabalho. As dificuldades encontradas pelos estudantes iniciam-se a partir do momento da escolha da universidade; onde ficar para realizar a prova de vestibular e após o vestibular, na confirmação do resultado surgem as questões de moradia fixa, locomoção, alimentação e gastos.

Atualmente as vagas de moradia são divulgadas nos painéis das universidades, sendo necessário o estudante já estar na cidade e deslocar-se até a universidade para verificar as vagas e demais dúvidas em relação a imobiliárias, transporte, etc. Se o estudante já conhece outro estudante e/ou parente que já reside na cidade escolhida, as dificuldades são resolvidas mais facilmente, caso contrário gera-se desorientação, incertezas, gastos desnecessários, correrias. Com o desenvolvimento deste trabalho, o objetivo é desenvolver uma ferramenta para a centralização destas informações, com a colaboração dos usuários para o conteúdo da rede social, ou seja, um *software* colaborativo que é um “sistema baseado em computador que auxilia grupos de pessoas envolvidas em tarefas comuns (ou objetivos) e que provê interface para um ambiente compartilhado". (SOFTWARE COLABORATIVO, 2011).

As atuais ferramentas sociais disponíveis na *Web* não possuem o enfoque do problema citado, havendo assim, informações espalhadas pelo site sem o devido filtro de conteúdo, sem organização por categorias, pouco detalhamento de serviços

emergenciais e serviços de transporte público, ou seja, serviços essenciais para prover toda a infra estrutura necessária para o novo morador na cidade escolhida.

3.2 CICLO DE VIDA DO OBJETO SOCIAL

Como já citado neste trabalho, na seção 2.3, os sites de redes sociais oferecem uma complexa variedade de opções para gerenciamento de atividades. Características que todos possuem em comum são: perfil de usuário e algum tipo de ferramenta para comunicação em grupo.

O início de cada rede social é uma ideia útil e relativamente pequena, direcionada a um grupo de pessoas específico. Esse enfoque voltado para um grupo de pessoas ou uma atividade, é vital para dar senso de coesão e objetivo ao site (BELL, 2010). Neste trabalho, a interação entre os indivíduos na rede social será baseada em serviços para atender estes, que estudam e/ou trabalham fora de suas cidades natais, com mecanismos de busca de anúncios de serviços, busca de conteúdo em geral, comentários, *tags* em anúncios e mensagens individuais para outros usuários; mecanismos necessários para os usuários motivarem-se a participar e a compartilhar a rede social.

3.3 REQUISITOS DA REDE SOCIAL

Os requisitos parciais foram levantados a partir da verificação das funcionalidades existentes em sites similares, apontando possíveis implementações na rede social, sendo verificadas também, quais as funcionalidades são inexistentes ou parcialmente desenvolvidas. Os sites são estes: www.easyquarto.com, www.dividir-apartamentos.com.br, dividir-apartamento.vivastreet.com.br. Os requisitos parciais levantados foram:

- ➔ Cadastro com foto de usuários;
- ➔ Fórum para discussão de assuntos relacionados ao universo estudantil;
- ➔ Postagem de tópicos somente por usuários cadastrados;
- ➔ Anúncios de vagas somente por usuários cadastrados;
- ➔ Visualização de tópicos por todos os usuários;
- ➔ Comentários nos tópicos (com filtragem de conteúdo);

- ➔ Postagem de fotos da república;
- ➔ Links de horários de ônibus, empresas de transporte;
- ➔ Links de imobiliárias com informações e anúncios de imóveis;
- ➔ Links de hotéis e respectivos preços;
- ➔ Links de entretenimento, festas e baladas;
- ➔ Links de cinema, quais filmes estão em cartaz;
- ➔ Links úteis: bombeiros, polícia, farmácias, taxis, supermercados;
- ➔ Links com descontos para estudantes;
- ➔ Todos os links com a localização em mapa;
- ➔ Integração com outras redes sociais (Orkut, Twitter, Facebook, Last.fm);
- ➔ Tópicos para encontros entre estudantes para reuniões pessoalmente;

São requisitos parciais, pois ao serem levantados, logo se notou a necessidade de um sistema eficiente de busca para englobar grande parte destes requisitos e gerenciar todas as informações de anúncios na rede social proposta.

4 BUSCA

Devido ao grande volume de informações geradas de forma colaborativa pelos usuários, a rede social deverá manter estruturas de dados flexíveis para gerenciar, auto-organizar e disponibilizar estas informações de maneira simples e prática. Em virtude de suas complexidades, um mecanismo de busca em toda rede social merece destaque, pois não exibirá apenas listas de puro texto e sim, resultados significativos de acordo com os critérios buscados (BELL, 2010).

A busca é um dos primeiros lugares comuns para novos visitantes explorarem em um site. Sendo esta uma premissa importante, deve-se criar uma interface de busca robusta, confiável e de bom desempenho, podendo ser diferenciada para cada membro e não membros da rede social (BELL, 2010).

Para estimar o volume de requisições de busca, foi definido o tamanho do grupo de usuários em potencial da rede social. Foram realizados estudos acerca do desempenho de três motores de busca, que estão descritos na seção 4.2, o operador padrão "*like*" utilizado na linguagem *SQL* (para interface com o banco de dados); índices do tipo *full-text* e a biblioteca *Lucene* mantida pelo *Apache Software Foundation* que implementa a *full-text search* (pesquisa integral).

Após esta etapa, iniciou-se o desenvolvimento do protótipo, implementando os três motores de busca estudados; a geração de massa de dados e testes de desempenho para validar os resultados das requisições de busca feitas para cada tecnologia de busca que estão descritos na seção 5.

4.1 DEFININDO GRUPO DE USUÁRIOS EM POTENCIAL

Para definir o grupo de usuário em potencial abrangendo o Brasil todo, foram consultados dados no censo do ano 2000 (IBGE, 2003), tendo-se aproximadamente 27,9% de estudantes que decidem estudar fora de suas cidades, sendo que, em 2009 (FERREIRA L., 2010), tem-se aproximadamente 6,5 milhões de estudantes universitários; Dentre 6,5 milhões de estudantes, 27,9% estudam fora, temos um total de 1.813.500 de estudantes.

Estimando quantos destes estudantes utilizam alguma ferramenta online para encontrar anúncios de moradia, temos aproximadamente 13% (FERREIRA L., 2010); sendo 13% de 1.813.500 estudantes, são em torno de 235.755 potenciais

usuários da rede social proposta neste trabalho. No Gráfico 1 está representado o grupo de usuários em potencial em torno dos estudantes que estudam fora.

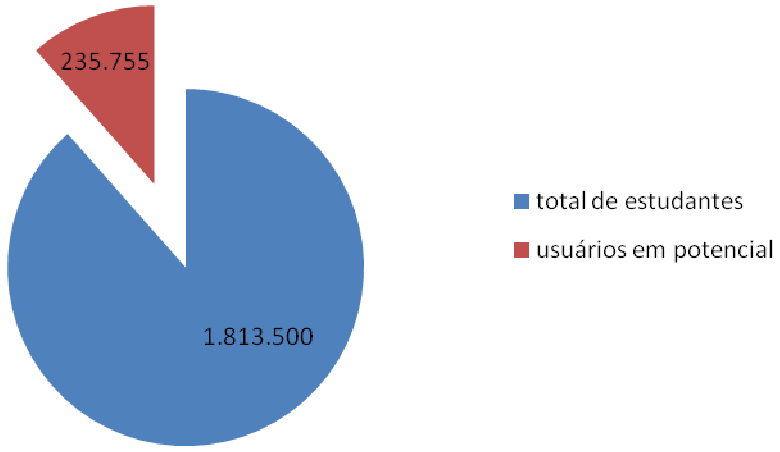


Gráfico 1 - Usuários em potencial
Fonte: Autoria Própria

4.2 TECNOLOGIAS PARA PESQUISA

Nas próximas seções estão descritas as tecnologias *SQL LIKE*, *SQL full-text* e *Lucene*, com um breve histórico, com características e basicamente como funcionam. Estas tecnologias foram escolhidas para serem apresentadas e confrontadas o seu desempenho como motor de busca.

4.2.1 Consultas *SQL* Utilizando *LIKE*

O operador lógico *LIKE*, introduzido no padrão *SQL92*, é um poderoso artifício para comparação de textos em bancos de dados relacionais. É usado em uma cláusula *where* para procurar um padrão em uma coluna. Com o operador podem ser utilizados os caracteres coringa, “%” (percentagem) e “_” (underline), o “%” indica que um ou mais caracteres podem aparecer no texto e o “_” representa um único caractere qualquer no texto. É suportado por todos os SGBDs que utilizem *SQL*. É uma causa comum de problemas de desempenho, dependendo do termo de pesquisa (HJORT, 2008).

No artigo escrito por Rodrigo Hjort para a revista SQL Magazine 2008, são abordadas técnicas para garantir um bom desempenho no *PostgreSQL* ao utilizar o operador *LIKE*. Foi utilizada uma base com 10,5 milhões de registros. Primeiro caso – “%” ao final, expressão usada ‘*SELECT * FROM PESSOAS WHERE NOME LIKE*

‘*JOSE CARLOS %*’’; Inevitavelmente uma varredura seqüencial (*full scan* ou *sequential scan*) na tabela foi invocada, levando cerca de 5 segundos para exibir os resultados. Uma varredura sequencial em um banco de dados com milhões de registros é extremamente danosa, foi criado um índice na tabela pessoas para o campo nome. A busca indexada levou 138 milissegundos, contra 4,77 segundos da busca sequencia, retornando 31 mil registros, com a ajuda do índice, o tempo de resposta ficou em menos de 3 % do tempo da busca original. Segundo caso – “%” no início, expressão usada ‘*SELECT * FROM PESSOAS WHERE NOME LIKE ‘% DOS SANTOS FILHO”*; Consulta demorou mais de 6,5 segundos. Pois o “%” invalidou o índice anteriormente criado. Para correção, é necessário criar um índice reverso (inverte os *bytes* da chave a ser indexada, permitindo que o último caractere da chave, espalhe os valores entre as estruturas de dados no índice, aumentando consideravelmente o desempenho no momento da busca). No banco de dados de testes foram retornadas mais de 2 mil linhas, e a busca indexada reversa levou 73 milissegundos, contra 6,54 segundos da busca sequencial. Com isso, a busca com o índice reverso levou cerca de 1% da busca original. Terceiro caso – “%” em ambas as extremidades, expressão usada *‘SELECT * FROM PESSOAS WHERE NOME LIKE ‘%JOSE CARLOS % SANTOS FILHO%”,* neste caso, infelizmente não há como otimizar. Haverá varredura sequencial na tabela, mesmo utilizando o índice reverso. Na Tabela 1 está o comparativo dos resultados e em negrito estão destacadas as situações em que a presença do índice no banco de dados torna-se interessante, com redução do tempo de resposta.

Tabela 1 - Tempos de resposta obtidos em cada abordagem de índice e expressão com o LIKE
Fonte: HJORT, 2008

Abordagem	NOME%	%NOME	NO%ME	%NO%ME%
sem índice	6,15 s	6,14 s	6,13 s	6,15 s
índice regular	**84,7 ms**	6,61 s	**85,0 ms**	5,65 s
índice reverso	149 s	**78,9 ms**	**65,2 ms**	148 s

Resultados demonstram que o operador *LIKE*, pode não ser suficiente em determinados casos, sendo interessante o investimento em outras abordagens para

pesquisas textuais, tal como *Full-Text Indexing* (indexação integral de texto) e *Full-Text Search* (pesquisa integral) (HJORT, 2008).

4.2.2 Criação de Índices no *MySQL*

A otimização de banco de dados nem sempre é simples e nem sempre garantida. Quando uma tabela não possui índices, os seus registros são desordenados e uma consulta terá que percorrer todos os registros, efetuando uma varredura sequencial. Se houver criação de um índice, uma nova tabela é gerada. A quantidade de registros da tabela de índices é o mesmo para a tabela original, com a diferença que os registros são ordenados, implicando na “varredura” de uma consulta, onde ela é cessada quando um valor imediatamente maior é encontrado. Os índices devem ser criados em colunas que serão usadas para pesquisa, ordenação ou agrupamento (FERREIRA T., 2010).

Existem desvantagens de tempo e espaço em disco, pois os índices aumentam a velocidade de consultas, mas diminuem a velocidade de inserções, atualizações e exclusões, ou seja, índices tornam mais lentas as operações de escrita, pois ao escrever um registro, a ordenação dos índices é alterada. Quanto mais índices houver numa tabela, mais mudanças de ordenação necessitam ser feitas, degradando o desempenho. Em segundo lugar, um índice ocupa espaço de disco. Em tabela *MyISAM*, por exemplo, os dados são mantidos em um arquivo de dados e os índices são mantidos em arquivos de índices. Cada índice do arquivo de índices consiste de um conjunto de chaves de registros ordenados que possam ser usados para localizar dados de forma rápida (FERREIRA T., 2010).

A partir da versão 3.23.23, o *MySQL* tem suporte para indexação e busca *full-text*. Os índices *full-text* são um índice do tipo *FULLTEXT* e são usados apenas com tabelas *MyISAM*, podendo ser criadas a partir de colunas *CHAR*, *VARCHAR* ou *TEXT* durante um *CREATE TABLE* ou adicionados posteriormente com *ALTER TABLE* ou *CREATE INDEX* (Manual de Referência do MySQL 4.1. Pesquisa Full-text no MySQL, 2010). É possível encontrar implementações de *full-text search* nos SGBDs *PostgreSQL* (PostgreSQL, 2011) e *SQLServer* (MSDN, 2011).

4.2.2.1 Índices full-text no MySQL

Diferentemente de quando é necessário buscar valores exatos utilizando o sinal de igualdade ou ainda, buscar *strings* que comecem, terminem ou contenham

uma determinada *substring* através de alguma estrutura provida pela linguagem *SQL*; o *full-text search* é um mecanismo que nos permite procurar termos específicos no formato *string* em meio a grandes quantidades de texto que estejam localizados em colunas de tabelas de um banco de dados. Estes textos, ao serem cadastrados em tais colunas, terão cada uma de suas palavras indexadas por índices chamados *FULLTEXT INDEX*, que também são parte do mecanismo que nos possibilita realizar a busca (BIANCHI, 2009).

Ao criar tabelas em um banco de dados *MySQL* e tendo em mente que haverá na interface do sistema um campo para busca por termos, palavras e/ou frases, baseada em uma coluna de uma tabela que armazena grande quantidade de texto, podemos utilizar nestas colunas índices do tipo *FULLTEXT*, sendo possível a utilização também das funções disponíveis *MATCH()* e *AGAINST().* Existem três tipos de buscas por termos no *MySQL* (BIANCHI, 2009):

- *Natural Language Full-text Searches*

  A pesquisa *full-text* é realizada com a função *MATCH().*

  *SELECT * FROM tabela WHERE MATCH (índices full text) AGAINST ('string pesquisada');*

  A função *MATCH()* realiza uma busca de linguagem natural por uma *string* contra uma coleção de texto (um conjunto de uma ou mais colunas incluídas em um índice *FULLTEXT*. A *string* pesquisada é dada como o argumento de *AGAINST()*, a busca é realizada na forma caso-insensitivo. Para cada uma das linhas da tabela, *MATCH()* retorna um valor relevante, isto é, uma medida de similaridade entre a *string* pesquisada e o texto naquela nas colunas identificadas na lista *MATCH()*. Quando *MATCH()* é utilizado na claúsula *WHERE*, as linhas retornadas são automaticamente ordenadas com a maior relevância primeiro (Manual de Referência do MySQL 4.1. Pesquisa Full-text no MySQL, 2010).

- *Full-text Search With Query Expansion*

  A partir da versão 4.11, pesquisa *full-text* suportam expansão de consulta("*blind query expansion*"), que é útil quando a frase pesquisada é muito curta, o que normalmente significa que um usuário está confiando em um conhecimento contido, que a pesquisa *full-text* normalmente perde. Por exemplo, um usuário está pesquisando por "database" podem realmente significar que "MySQL", "Oracle", "DB2", "RDBMS" são todas frases que

devem coincidir com “databases" e devem ser encontrados também; utiliza-se desta forma: *SELECT * FROM* tabela *WHERE MATCH (*índices *full text) AGAINST (‘string* pesquisada’ *WITH QUERY EXPANSION)* (Manual de Referência do MySQL 4.1. Pesquisa Full-text no MySQL, 2010).

- *Boolean Full-text Search*

  Também é possível realizar buscas *full-text* booleanas, usando o modificador *IN BOOLEAN MODE*, utilizando operadores como + (a palavra deve estar em cada linha retornada), - (a palavra não deve estar em cada linha retornada), entre outros. E os registros pesquisados não são ordenados automaticamente em ordem decrescente de relevância (Manual de Referência do MySQL 4.1. Pesquisa Full-text no MySQL, 2010).

O *MySQL* utiliza um analizador muito simples que separa o texto em palavras, qualquer palavra que esteja na lista *stopwords* (palavras de parada), e se repita em pelo menos 50% das linhas de uma tabela, ou for menor que o padrão é ignorada. O tamanho padrão mínimo das palavras que serão encontradas pela pesquisa *full-text* é de quatro caracteres (Manual de Referência do MySQL 4.1. Pesquisa Full-text no MySQL, 2010).

Os valores de relevância são computados obtendo-se o número de palavras que compõem a linha, o número de palavras únicas desta linha, o número total de uma coleção de palavras e o por fim, o número de linhas que contém uma palavra em particular. São valores ponto flutuantes positivos. Deste modo, uma palavra que está presente em vários documentos, terá peso menor (e poderá ter até mesmo um peso zero), já que ele possui um valor semântico baixo nesta coleção. Por outro lado, se a palavra é rara, ela receberá um peso alto. O peso das palavras são então combinados para computar a relevância das linhas. Tal técnica funciona melhor com coleções grandes. Uma palavra que está presente em mais da metade das linhas, não produzirá resultados, pois ela é efetivamente tratada como *stopword* (palavra de parada, uma palavra com valor semântico zero). Uma palavra que apareça repetidas vezes nos registros de uma tela, tem menos chance de encontrar documentos relevantes, sendo mais provável exibir resultados irrelevantes (Manual de Referência do MySQL 4.1. Pesquisa Full-text no MySQL, 2010).

4.3 LUCENE

Lucene é uma biblioteca de recuperação de informação textual (do inglês "*information retrieval*") de código aberto e criado por Doug Cutting em 1997. Originalmente, foi escrita em Java, mas foi rapidamente adaptada a outras linguagens de programação. Em 2004 foi integrada ao projeto *Apache*. Possui comunidade ativa de desenvolvedores. Por se tratar de uma API, o *Lucene* é composto por uma série de funções que possibilita a indexação e a busca de arquivos de diversos formatos, mas essas funções somente são acessíveis através do desenvolvimento de aplicações, pois o *Lucene* não dispõe de uma interface gráfica pronta (GOSPODNETIC, HATCHER e MCCANDLESS, 2010). Oferece escalabilidade (habilidade de manipular uma porção crescente de trabalho de forma uniforme), alto desempenho, pesquisas baseadas em palavras-chave, poderosas e ranqueadas (os resultados mais relevantes aparecem primeiro), permite atualização e busca simultânea, suporta diversas fontes de dados (email, páginas *Web*, arquivos, banco de dados) que possam ser transformados em texto; pois trabalha com palavras em um índice invertido. A Biblioteca é composta por duas partes principais: indexação (descrita na seção 4.3.1) e pesquisa (descrita na seção 4.3.2).

4.3.1 Indexação

A indexação, basicamente, consiste de quatro passos:

1. Coletar os dados. **2.** Processar o texto (análise). **3.** Criar documentos **4.** Indexar (armazenar os documentos no índice).

O primeiro passo na indexação de dados é coletá-los e convertê-los em um formato de texto simples. Podendo ser utilizado conversores de dados personalizados. No segundo passo, é feito uma análise no texto que foi extraído anteriormente, para encontrar informações não relevantes que não precisam ser incorporadas aos índices, utilizando a classe *Analyzer* (VELOSO, 2008).

4.3.1.1 Classe *Analyzer*

Análise é a conversão dos dados de texto em uma unidade de procura fundamental, chamada de "termo". Durante a análise, os dados de texto passam por várias operações: extração das palavras, remoção de palavras comuns (*stopwords*), descartar pontuação, reduzir palavras a uma forma de raiz (*stemming*), alteração das palavras para minúsculas, etc. Um analisador, uma instância da classe abstrata

Analyzer, é um encapsulamento do processo de análise, no qual as palavras do texto são separadas e convertidas em *tokens*, e estes *tokens* são incluídos como termos no índice do *Lucene* (GOSPODNETIC, HATCHER e MCCANDLESS, 2010).

O *Lucene* é fornecido com vários analisadores integrados. Eles diferem na maneira pela qual analisam o texto e aplicam os filtros. Conforme a análise remove as palavras antes de indexar, ela diminui o tamanho do índice, mas isso poderá ter um efeito negativo na precisão do processamento da consulta. É possível ter maior controle sobre o processo de análise ao criar analisadores customizados usando os blocos de construção básicos fornecidos pelo *Lucene* (GOSPODNETIC, HATCHER e MCCANDLESS, 2010).

No terceiro passo, é criado o documento (*Document*) e os seus respectivos campos (*Field*) que serão indexados. A classe *Field* engloba o nome do campo e seu valor. O *Lucene* fornece opções para especificar se um campo precisa ser indexado ou analisado e se o valor precisa ser armazenado. Essas opções podem ser transmitidas ao criar uma instância de *Field*. Na tabela 2 estão descritas as opções de metadados de *Field* (SONAWANE, 2009).

Tabela 2 – Opções de indexação de *Fields*
Fonte: Adaptado de Sonawane, 2009.

Opção	**Descrição**
Field.Store.Yes	Usado para armazenar o valor dos campos. Ideal para campos exibidos com o caminho de arquivo de resultados de procura — e com a URL, por exemplo.
Field.Store.No	O valor de campo não é armazenado — o corpo da mensagem de e-mail, por exemplo.
Field.Index.No	Ideal para campos pouco procurados— usados com os campos armazenados, como por exemplo, caminho de arquivo.
Field.Index.ANALYZED	Usado para campos indexados e analisados (passam pela classe *Analyzed*, serão tokenizados) — o corpo e o assunto da mensagem de e-mail, por exemplo.
Field.Index.NOT_ANALYZED	Usado para campos indexados, mas não

	analisados Ele preserva o valor original do campo em sua totalidade — datas e nomes pessoais, por exemplo.

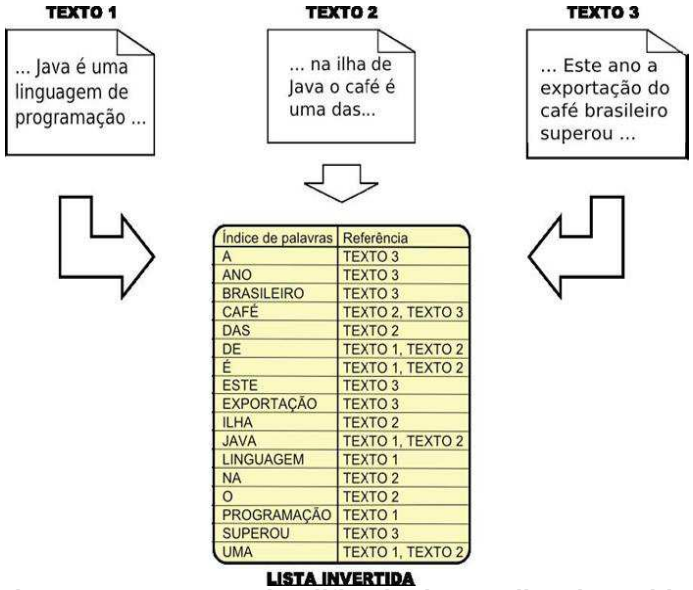
No quarto passo, determina-se uma localização física para o índice. Esta localização física é encapsulada em uma classe abstrata, a *Directory*, sendo esta classe responsável em indicar a localização dos índices a serem pesquisados ou onde os mesmos deverão ser criados. Cria-se uma instância da classe *IndexWriter* que fará a indexação dos Documentos.

4.3.1.2 IndexWriter

É o componente central do processo de indexação. Essa classe cria um novo índice ou abre um já existente, e adiciona, remove ou atualiza documentos no índice. Não permite a leitura e a pesquisa do índice (GOSPODNETIC, HATCHER e MCCANDLESS, 2010).

4.3.1.3 Índice Invertido

O conceito por trás do índice invertido é análogo a um índice no final de um livro, permitindo localizar rapidamente páginas que falam sobre determinados temas. No caso do *Lucene*, um índice é uma estrutura de dados inter-relacionada (como uma lista invertida), normalmente armazenados no sistema de arquivos como um conjunto de chaves que apontam para os documentos nos quais ocorre, permitindo acesso rápido aleatório para palavras armazenadas dentro dele. Evitando assim, a varredura sequencial no arquivo em busca da palavra ou frase desejada, que peca na confiabilidade dos dados exibidos e no custo de desempenho para a execução da pesquisa. A utilização de índices na recuperação de informações contribui para minimizar o problema dessa abordagem. Na Figura 2 está representado o esquema de uma lista invertida (VELOSO, 2008).



Índice de palavras	Referência
A	TEXTO 3
ANO	TEXTO 3
BRASILEIRO	TEXTO 3
CAFÉ	TEXTO 2, TEXTO 3
DAS	TEXTO 2
DE	TEXTO 1, TEXTO 2
É	TEXTO 1, TEXTO 2
ESTE	TEXTO 3
EXPORTAÇÃO	TEXTO 3
ILHA	TEXTO 2
JAVA	TEXTO 1, TEXTO 2
LINGUAGEM	TEXTO 1
NA	TEXTO 2
O	TEXTO 2
PROGRAMAÇÃO	TEXTO 1
SUPEROU	TEXTO 3
UMA	TEXTO 1, TEXTO 2

Figura 1 - Esquema simplificado de uma lista invertida.
Fonte: VELOSO, 2008

4.3.2 Pesquisa

A pesquisa, por sua vez, obtém a expressão de consulta informada pelo usuário, pesquisa no índice e organiza os resultados pela similaridade do texto com a consulta. A pesquisa ao índice ocorre após instanciar a classe abstrata *IndexSearcher*, que abre o índice em modo somente leitura, informando como parâmetro a localização do *Directory* (onde o índice está armazenado). Os critérios de busca são passados para a função de busca do *Searcher* através do Objeto *Query*.

4.3.2.1 Query

As implementações concretas da classe abstrata *Query* definem diferentes formas de pesquisa textual, como pesquisa por palavra (*TermQuery*), por frase (*PhraseQuery*), dentro de um intervalo de termos (*TermRangeQuery*), por termos que iniciam com prefixo (*PrefixQuery*), com cláusulas elaboradas (*BooleanQuery*), com caractere coringa (*WildcardQuery*) e com palavras semelhantes ao termo pesquisado (*FuzzyQuery*), entre outros (GOSPODNETIC, HATCHER e MCCANDLESS, 2010).

Em seguida, o objeto *Query* é construído através de uma instância da classe escolhida para realizar a busca, para o qual é passado como parâmetro o nome do campo a ser procurado e o valor que possivelmente ele contém. As consultas através destas classes não requerem um analisador e se baseiam na

correspondência dos termos com o que foi indexado. Portanto, para construir consultas utlizando estas classes, é necessário garantir que os termos incluídos em todas as suas consultas sejam correspondentes aos *tokens* produzidos pelo analisador utilizado durante a indexação. Porém, existe a classe *QueryParse*, que é capaz de construir o objeto *Query* apropriado a partir da expressão de pesquisa, utiliza analisador, que deve ser o mesmo utilizado no processo de indexação, e engloba as subclasses de implementação concreta da classe *Query* (GOSPODNETIC, HATCHER e MCCANDLESS, 2010).

O resultado da consulta é um conjunto de documentos ordenado por relevância, ou seja, os itens mais similares a consulta aparecem primeiro. O *Lucene* tem um mecanismo de atribuição de relevância, baseado no TF-IDF, que é uma medida estatística que dá mais peso a documentos com maior frequência do termo, e a termos pouco comuns no geral (RODRIGUES, 2008).

As classes primárias envolvidas na recuperação dos resultados de procura são *ScoreDoc* e *TopDocs*.

- *ScoreDoc* - Um ponteiro simples para um documento contido nos resultados de procura. Isso engloba a posição de um documento no índice e a pontuação calculada pelo *Lucene*.
- *TopDocs* - Engloba o número total de resultados de procura e uma matriz de *ScoreDoc*.

A pesquisa oferece diversas opções de classificação, filtragem e ordenação dos resultados da pesquisa (GOSPODNETIC, HATCHER e MCCANDLESS, 2010).

4.3.3 *Powered By Lucene*

Existe uma série de projetos que utilizam o *Lucene* como biblioteca de pesquisa, como por exemplo:

- *Salesforce.com* oferece diversos aplicativos (vendas, atendimento ao cliente e colaboração) rodando na nuvem, utiliza *Lucene* para suportar buscas com diferentes configurações de privacidade, indexação e atualização de dados estruturados e não estruturados. (Apache Lucene Eurocon Barcelona, 2011)
- *Etsy.com* é uma comunidade *online* que comercializa itens antigos e de artesanato, utiliza *Lucene* para servir consultas a uma taxa de mais de 8 bilhões por ano, com facetamento, filtragem das consultas, valores de moeda

atualizados e sugestão de itens através da localização geográfica através do *IP* do usuário. (Lucene & Solr User Conference, 2011)

- *Twitter.com* é uma rede social, utiliza *Lucene* para suportar pesquisas textuais, imagens e vídeo em tempo real em torno de 1.5 bilhões de requisições por dia e atualizar o índice com 200 milhões de *tweets* por dia. (Apache Lucene Eurocon Barcelona, 2011)
- *Cisco Pulse*™ é uma plataforma disponibilizada pela Cisco, utiliza *Lucene* para indexar, buscar e atualizar palavras chaves e tópicos de várias formas de mídia, desde email até vídeo, em uma rede interna. (Lucene & Solr User Conference, 2011)

4.3.4 Estudo de caso *LinkedIn* com *Lucene*

LinkedIn.com é a maior rede social para profissionais no mundo, com mais de 60 milhões de usuários na *Web* (Março de 2010), e onde os usuários do site possuem um perfil onde publicam seus resumos profissionais ou curriculum vitae. A busca por pessoas no *LinkedIn* é complexa, integrando uma arquitetura distribuída, indexação em tempo real e busca personalizada. Cada usuário possui uma rede de contatos que influencia os resultados da busca. *LinkedIn* usa duas extensões integradas com *Lucene*, uma é *Bobo Browse* que provê informação facetada em cada busca e *Zoie* que provê um sistema de busca em tempo real (GOSPODNETIC, HATCHER e MCCANDLESS, 2010).

Bobo Browse é um sistema que oferece facetamento de busca para *LinkedIn*, sem necessidade de reindexar o índice. Facetamento que facilita o refinamento da busca, filtrando a busca por pessoas, por exemplo, o filtro pode ser por local atual de trabalho, por localização, por companhias em quais já trabalhou etc. (GOSPODNETIC, HATCHER e MCCANDLESS, 2010).

Zoie é um sistema *open source* de indexação em tempo real, é usado como um *cluster* de busca atendendo a requisições de pessoas, trabalhos, companhias, notícias, grupos, fóruns e muito mais. Para a busca de pessoas, *Zoie* (*IS DEPLOYED*) de modo distribuído e (*IS SERVING*) mais de 50 milhões de documentos em tempo real. Cada servidor instancia duas JVMs, cada com uma instância de *Zoie* gerenciando cerca de 5 milhões de requisições por dia, cada servidor possui uma média de latência de apenas 50 milissegundos, enquanto processa cerca de 150.000 atualizações por dia. Percebe-se que *Zoie* é um sistema

poderoso para soluções de buscas em tempo real (GOSPODNETIC, HATCHER e MCCANDLESS, 2010).

5 DESENVOLVIMENTO

Nas seções seguintes são apresentados a quantidade necessária de massa de dados de anúncios que deve ser gerada e como esta massa de dados foi gerada. O desenvolvimento do protótipo, com quais ferramentas foi desenvolvido, são apresentadas figuras com os esquemas de cada classe criada, diagramas de sequencia e explicação dos métodos de cada classe.

5.1 REGRA 90:9:1

Conforme já citado neste trabalho na seção 2.3.1, a regra 90:9:1 foi definida como “desigualdade de participação” (NIELSEN, 2006), onde a participação dos usuários divide-se desta maneira: sendo 90% (*lurkers,* espreitadores) apenas acessam o conteúdo (lendo e observando, sem contribuir na comunidade), 9% (*editors,* editores) contribuem de vez em quando (modificando ou adicionando informações) e 1% (*creators,* criadores) contribui de forma ativa e constante (criadores de conteúdo), conforme ilustrado na Figura 2.

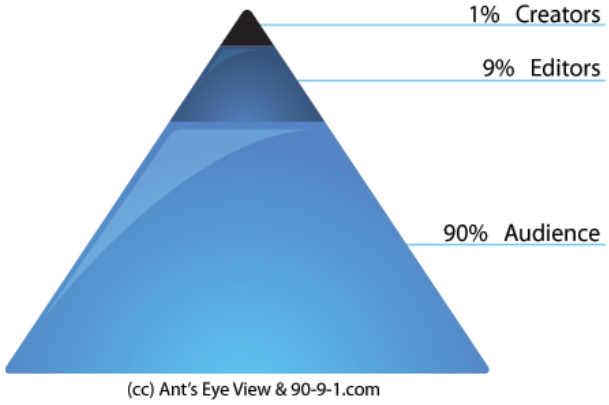


Figura 2 - Relação 90:9:1
Fonte: NIELSEN, 2006

Aplicando esta regra no grupo de usuários em potencial que foi definido na seção 4.1, que são 235.755 usuários; 1% de conteúdo novo = 2.357 + 9 % de conteúdo modificado = 21.217, necessita-se de 23.574 anúncios para efetuar os testes, sendo este valor, uma média de acessos mensais da rede social.

5.2 MASSA DE DADOS

Sendo necessário gerar em torno de 23.500 anúncios, foi utilizada uma ferramenta *open source* de geração de massa de dados, a *generatedata.com*, que

gera dados de teste em *HTML, EXCEL, SQL, XML*; utilizando *Lorem Ipsum* (é um texto padrão em latim, utilizado para testes). Sendo este um impeditivo, pois o texto gerado não atendeu as necessidades, pois precisamos de texto em português. Após a instalação do programa foi possível a alteração do texto, com um arquivo de texto contendo anúncios copiados aleatoriamente dos sites www.easyquarto.com, www.dividir-apartamentos.com.br, dividir-apartamento.vivastreet.com.br. Foram gerados 20.000 anúncios.

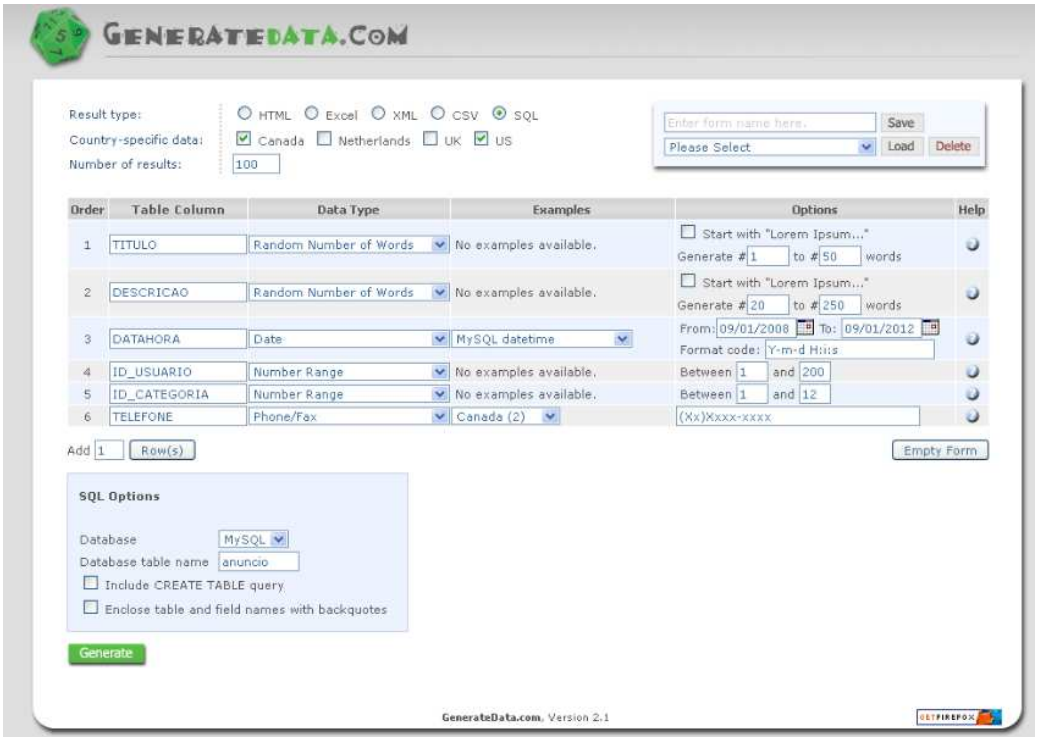


Figura 3 – Configuração para gerar anúncios no generatedata.com
Fonte: Autoria Própria

A saída do *generatedata.com* é uma instrução *SQL* de *INSERT*, como o exemplo: *INSERT INTO anuncio (TITULO,DESCRICAO,DATAHORA,ID_USUARIO,ID_CATEGORIA,TELEFONE) VALUES ('*Gávea, a 3 minutos da PUC, que seja tranquilo, organizado, responsável e com hábitos saudáveis. Dou preferência','individuais ou em conjunto (até quatro pessoas no mesmo quarto). Casa segura, ótima localização (perto de academia, padaria, mercados e clínicas). Praticamente no Centro de Camaçari. Alugo apartamento ou quartos no Calçadão de Caxambu, próximo ao Parque das Águas. Apenas por curto período como feriados, férias, finais de semana, congressos, etc. O','2011-04-18 17:38:29','103','7','(49)8933-1056');

5.3 PROTÓTIPO

O Código fonte original de indexação e pesquisa foi desenvolvido por Amol Sonawane e foi disponibilizado para elaboração de artigo Usando o *Apache Lucene* para Procura de Texto (SONAWANE, 2009).

O Código fonte original do analisador em português foi desenvolvido por Sérgio Veloso e foi disponibilizado para elaboração de artigo Conhecendo o *Apache Lucene* (VELOSO, 2008).

Foi desenvolvido para a implementação do *Lucene*: o indexador, *servlet* que requisita a classe *bean* para fazer a consulta e retornar os dados para o *index.jsp*. Para a implementação do *SQL* e *SQL Full-text search*, foram implementados classes *beans* e classes que fazem a comunicação com o banco de dados. Posteriormente, serão realizados testes com estes *servlets* para avaliar o desempenho da pesquisa.

O protótipo foi desenvolvido utilizando as seguintes ferramentas:

- ***MySQL*** é um SGBD, que utiliza a linguagem *SQL* para manipulação de dados em RDBMS, sendo considerada um ferramenta de manipulação de base de dados de tamanho moderado. O *MySQL* foi criado na Suécia em meados de 1980. É atualmente um dos bancos de dados mais populares, com mais de 10 milhões de instalações pelo mundo (Redação Oficina da Net, 2010). O *MySQL* foi o banco de dados escolhido por apresentar extensa documentação, milhares de sites na internet, mas principalmente pela sua fácil instalação e integração com o servidor *Web* e ser distribuído gratuitamente. (Redação Oficina da Net, 2007).
- **Java** é uma linguagem de programação orientada a objetos, desenvolvida pela empresa *Sun Microsystems* e lançada em maio de 1995. Entre suas principais características, estão: a orientação a objetos, a portabilidade, vasto conjunto de bibliotecas e APIs e código aberto (JAVA, 2011). Esta linguagem foi escolhida devido à familiarização com seus recursos devido à utilização nas disciplinas deste curso de graduação e por sugestão do orientador.
- **NetBeans** é uma IDE *open source*. Desenvolvido pela *Sun Microsystem* em 2000. É uma ferramenta para programadores escreverem, compilarem, depurarem, implantarem e gerenciar projetos e arquivos. É escrito em Java - mas pode suportar qualquer linguagem de programação (NetBeans, 2011).
- **TomCat** é um servidor de aplicações Java para *Web*, um container *Web* que abrange tecnologias como *Servlet* e JSP. Possui o código de fonte aberto,

desenvolvido pela Fundação *Apache* no projeto *Apache Jakarta* (The Apache Software Foundation, 2011).

- **JSP** é uma tecnologia utilizada no desenvolvimento de aplicações para *Web* baseada na linguagem de programação Java, ou seja, é uma página HTML que contém código Java (TeHospedo, 2011).
- **JSTL** é a API que encapsulou em *tags* simples toda a funcionalidade que diversas páginas *Web* precisam, como controle de laços (*for*), controle de fluxo do tipo *if else*, manipulação de dados XML e a internacionalização de sua aplicação. É utilizada para padronização de páginas JSP (CAELUM).
- ***Servlet*** é uma classe em Java que processa requisições e respostas para a camada de apresentação de um aplicativo *Web*, estendendo a funcionalidade de um servidor *Web* (Softech, 2006).

5.3.1 Banco de Dados

O padrão da tabela *anuncio* que foi criada para armazenar os anúncios utilizados neste trabalho, é do tipo *InnoDB*. Na Tabela 3 apresentam-se os campos contidos na tabela *anuncio*:

Tabela 3 - Tabela anuncio
Fonte: Autoria Própria

CAMPO	TIPO
ID_ANUNCIO	int (PK)
TITULO	varchar (512)
DESCRICAO	varchar (2056)
ID_USUARIO	int
ID_CATEGORIA	int
TELEFONE	varchar (13)
DATAHORA	timestamp

Para a criação do índice *fulltext*, criou-se uma cópia da tabela *anuncio* para *anuncio_fulltext*. Alterou-se o padrão da tabela para *MyISAM*, utilizando a seguinte instrução *SQL*: *ALTER TABLE anuncio_fulltext ENGINE=MyISAM*. Após a alteração, criou-se o índice *fulltext* no campo DESCRICAO, utilizando a seguinte instrução *SQL*: *ALTER TABLE anuncio_fulltext ADD FULLTEXT INDEX (DESCRICAO)*.

5.3.2 Classe Indexador

Na Figura 4 está representado o esquema da classe indexador.

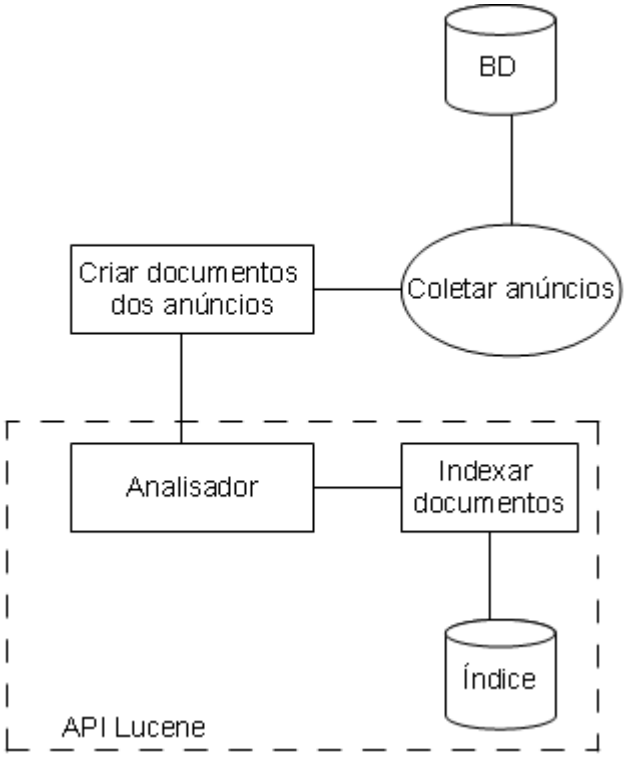


Figura 4 - Diagrama do funcionamento da classe Indexador
Fonte: Autoria Própria

Na Figura 5 está o diagrama de sequencia da classe Indexador.

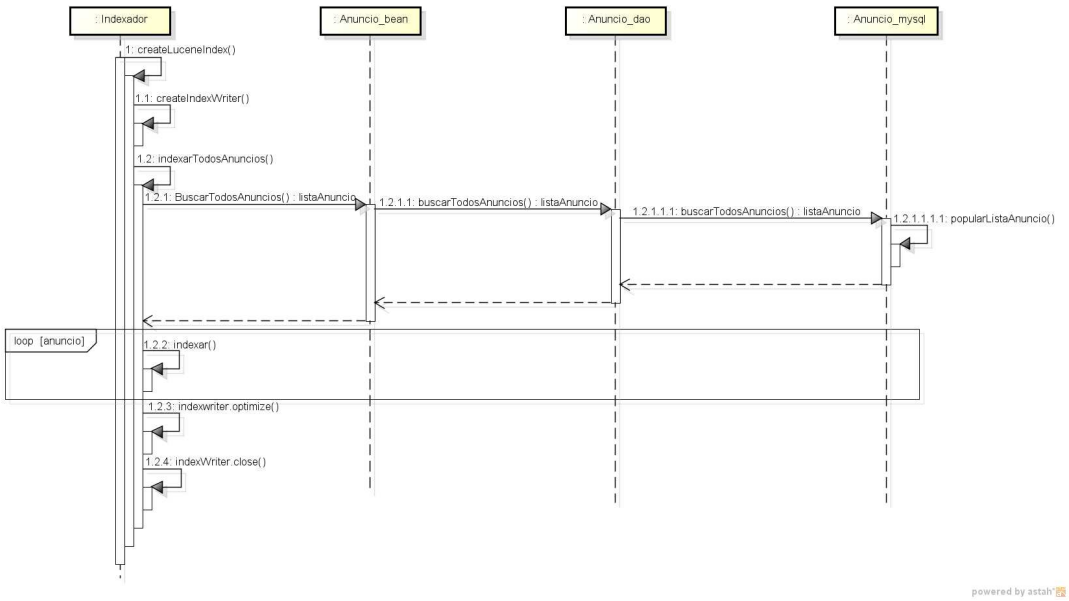


Figura 5 - Diagrama de sequencia da classe Indexador
Fonte: Autoria Própria

Na Figura 6 estão os métodos da classe Indexador.

```
1   package indexador;
2
3   import ...
19
20  public class Indexador {
21
22      private IndexWriter indexWriter;
23      private String indexDirectory, id_anuncio, titulo, descricao, datahora;
24
25      public static void main(String[] args) throws FileNotFoundException, IOException {...}
29
30      private void createLuceneIndex() {...}
43
44      void createIndexWriter() {...}
61
62      public void indexar(Anuncio_regra anuncio) throws FileNotFoundException, IOException {...}
82
83      public void indexarTodosAnuncios() throws FileNotFoundException, IOException {...}
96
97      Anuncio_regra criarNovoAnuncio() {...}
108 }
```

Figura 6 - Classe Indexador
Fonte: Autoria Própria

Na Figura 7 exibe-se o método *createLuceneIndex* onde faz-se uma chamada ao método *createIndexWriter* e são setadas duas *flags* para calcular o tempo em milissegundos do processo de indexação para indexar todos os anúncios.

```
30      private void createLuceneIndex() {
31          createIndexWriter();
32          try {
33              long start = new Date().getTime();
34              indexarTodosAnuncios();
35              long end = new Date().getTime();
36              System.out.println("Indexação levou " + (end - start) + " millissegundos");
37          } catch (FileNotFoundException e) {
38              throw new RuntimeException(e);
39          } catch (IOException e) {
40              throw new RuntimeException(e);
41          }
42      }
```

Figura 7 - Método *createLuceneIndex*
Fonte: Autoria Própria

Na Figura 8 exibe-se o método *createIndexWriter*, onde cria-se uma instância da classe *Directory* onde serão armazenados os documentos indexados. Cria-se uma instância da classe *PortuguesAnalyzer*. Cria-se o *indexWriter*, e aplica-se também a política de delete, que customiza a exclusão de consolidações obsoletas do diretório de índice. A política de exclusão padrão é *KeepOnlyLastCommitDeletionPolicy*, que mantém apenas as consolidações mais recentes e remove imediatamente todas as consolidações anteriores depois que a nova consolidação for feita.

```
45      /**
46       * Este método cria uma instância do IndexWriter que é usado para adicionar
47       * Documents e escrever índices no disco.
48       */
49      void createIndexWriter() {
50          indexDirectory = "D:\\Chris - documentos\\NetBeansProjects\\Anuncio\\src\\indice";
51          try {
52              //Cria instância do Directory onde os documentos indexados serão armazenados
53              Directory fsDirectory = FSDirectory.getDirectory(indexDirectory);
54              Analyzer standardAnalyzer = new PortuguesAnalyzer();
55              //Cria um novo index
56              boolean create = true;
57              //Cria a instância de política de delete
58              IndexDeletionPolicy deletionPolicy = new KeepOnlyLastCommitDeletionPolicy();
59              indexWriter = new IndexWriter(fsDirectory, standardAnalyzer, create,
60                      deletionPolicy, IndexWriter.MaxFieldLength.UNLIMITED);
61          } catch (IOException ie) {
62              System.out.println("Erro ao criar IndexWriter");
63              throw new RuntimeException(ie);
64          }
65      }
```

Figura 8 - Método *createIndexWriter*
Fonte: Autoria Própria

Na Figura 9 exibe-se o método *indexarTodosAnuncios*, onde solicita-se à busca de todos os anúncios, através da instância da classe Anuncio_bean, retornando uma lista de anúncios cadastrados no banco de dados. Percorre-se esta lista, passando o objeto anuncio para o método indexar, e após percorrer toda a lista, requisita-se a otimização do índice (operação que compacta o banco de dados de índice e acelera as consultas) e fecha-se o índice.

```
88      public void indexarTodosAnuncios() throws FileNotFoundException, IOException {
89          try {
90              List<Anuncio_regra> listaAnuncio = new Anuncio_bean().getBuscarTodosAnuncios();
91
92              for (int i = 0; i < listaAnuncio.size(); i++) {
93                  indexar(listaAnuncio.get(i));
94              }
95          } finally {
96              indexWriter.optimize();
97              indexWriter.close();
98          }
99
100     }
```

Figura 9 - Método *indexarTodosAnuncios*
Fonte: Autoria Própria

Na Figura 10 exibe-se o método indexar, onde as variáveis: id_anuncio, titulo, descricao e datahora, recebem os valores dos campos que serão armazenados no índice. Os campos do *Document* são configurados e cria-se um objeto *Document*, e adiciona-se este documento ao índice, neste momento, o documento passa pelo analisador.

```
66
67      public void indexar(Anuncio_regra anuncio) throws FileNotFoundException, IOException {
68
69          id_anuncio = Integer.toString(anuncio.getId_anuncio());
70          titulo = anuncio.getTitulo();
71          descricao = anuncio.getDescricao();
72          datahora = anuncio.getDatahora();
73
74          Field id_anuncioField = new Field("id_anuncio", id_anuncio, Field.Store.YES, Field.Index.NO);
75          Field tituloField = new Field("titulo", titulo, Field.Store.YES, Field.Index.ANALYZED);
76          Field descricaoField = new Field("descricao", descricao, Field.Store.YES, Field.Index.ANALYZED);
77          Field datahoraField = new Field("data", datahora, Field.Store.YES, Field.Index.NOT_ANALYZED);
78
79          Document doc = new Document();
80          doc.add(id_anuncioField);
81          doc.add(tituloField);
82          doc.add(descricaoField);
83          doc.add(datahoraField);
84          indexWriter.addDocument(doc);
85
86      }
```

Figura 10 - Método *indexar*
Fonte: Autoria Própria

Na Figura 11 exibe-se a classe *PortuguesAnalyzer* que é o analisador utilizado para indexação dos anúncios.

```
1   package indexador;
2
3   import ...
13
14  /**
15   * PortuguesAnalyzer original disponibilizado pela Java Magazine
16   */
17  public class PortuguesAnalyzer extends Analyzer {
18
19      public static final String[] STOP_WORDS = new String[]{"a", "ao", "aos",
20          "as", "cuja", "cujas", "cujo", "cujos", "da", "das", "de", "do",
21          "dos", "e", "em", "daquele", "daquela", "daqueles", "daquelas",
22          "dela", "dele", "deles", "desta", "deste", "essa", "essas", "esse",
23          "esses", "esta", "estas", "este", "estes", "um", "uma", "umas",
24          "uns", "com", "lhe", "lhes", "mas", "me", "na", "nas", "nem",
25          "nesse", "nesta", "neste", "no", "nos", "no", "o", "os", "para",
26          "perante", "pois", "por", "quais", "qual", "que", "quem", "quer",
27          "se", "seu", "seus", "sob", "sobre", "sua", "suas", "so", "tal",
28          "tanto", "tem", "tendo", "teu", "teus", "toda", "todas", "todo",
29          "todos", "tua", "tuas", "r", "eu", "oi", "além", "ser", "há", "é", "á"};
30
31      private Set _stopWords = new HashSet();
32
33      public PortuguesAnalyzer() {
34          this._stopWords = StopFilter.makeStopSet(STOP_WORDS);
35      }
36
37      @Override
38      public TokenStream tokenStream(String fieldName, Reader reader) {
39
40          // Passa o texto no reader pelo tokenizer default do Lucene,
41          // retornando um TokenStream ou seja, uma sequencia de tokens.
42          // -> separa palavras por espaco em branco (whitespace).
43          // -> separa palavras por caracteres de pontuacao.
44          // -> emenda palavras seguidas de pontuacao se nao houver espaco em segu
45          TokenStream result = new StandardTokenizer(reader);
46          // Passa o tokenstream por um conjunto de filtros, em sequencia
47          // Filtro padrao
48          result = new StandardFilter(result);
49          // Converte todas as maiusculas em minusculas
50          result = new LowerCaseFilter(result);
51          // Remove qualquer palavra identificada como "stop word"
52          result = new StopFilter(result, _stopWords);
53          return result;
54      }
55  }
```

Figura 11 - Classe *PortuguesAnalyzer*
Fonte: Autoria Própria

5.3.3 Classe Pesquisa

Na Figura 12 está o diagrama de sequencia da classe Pesquisa, envolvendo as três implementações dos motores de busca, *SQL*, *SQL-FULLTEXT* e *Lucene*.

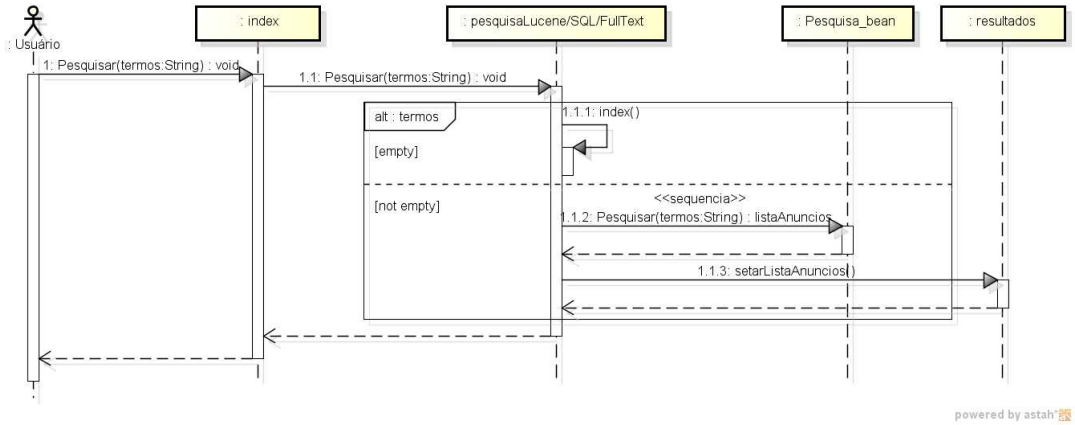


Figura 12 - Diagrama de sequencia da classe Pesquisa
Fonte: Autoria Própria

Na Figura 13 estão os métodos da classe Pesquisa.

```
1   package beans;
2
3   import ...
20
21  public class Pesquisa_bean {
22
23      private String indexDirectory = "D:\\Chris - documentos\\NetBeansProjects\\Anuncio\\src\\indice";
24      private Searcher indexSearcher;
25      private Pesquisa_regra pesquisa = new Pesquisa_regra();
26
27      public Pesquisa_bean() {...}
29
30      public void createIndexSearcher() throws CorruptIndexException, IOException {...}
33
34      private Anuncio_regra popularListaAnuncio(Document doc) {...}
42
43      public List<Anuncio_regra> showSearchResults(Query query) {...}
58
59      public List<Anuncio_regra> PesquisarLucene(String termos) throws org.apache.lucene.queryParser.ParseException {...}
68
69      public List<Anuncio_regra> PesquisarSQL(String termos) {...}
82
83      public List<Anuncio_regra> PesquisarSQLFullText(String termos) {...}
96
97      public Pesquisa_regra ResultadosPesquisa() {...}
100 }
```

Figura 13 - Classe Pesquisa
Fonte: Autoria Própria

5.3.3.1 Pesquisar SQL

Na Figura 14 está representado o esquema do método *PesquisarSQL*.

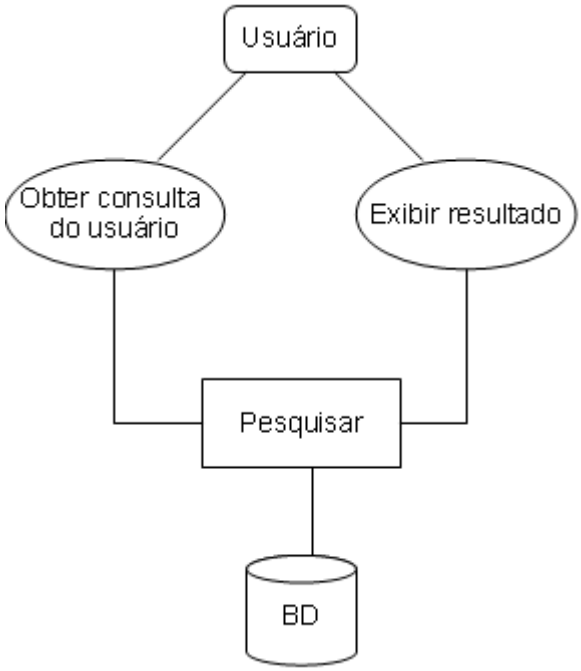


Figura 14 - Diagrama do funcionamento do método *PesquisarSQL*
Fonte: Autoria Própria

Na Figura 15 está o diagrama de sequencia do método *PesquisarSQL*.

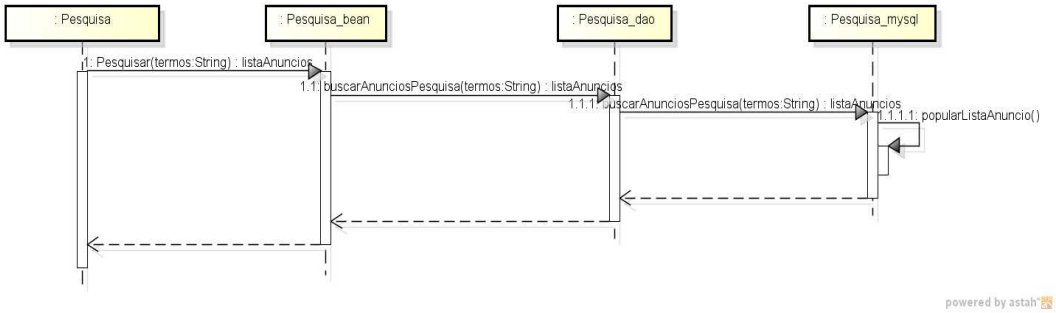


Figura 15 - Diagrama de sequencia do método *PesquisarSQL*
Fonte: Autoria Própria

Na Figura 16 exibe-se o método *PesquisarSQL* que ao fazer a busca passa-se como parâmetro a expressão de busca, e através da instância da classe Pesquisa_*dao*, retorna-se uma lista já populada na classe Pesquisa_*mysql*. Setam-se duas variáveis para verificar o tempo de busca em milissegundos.

```
71      public List<Anuncio_regra> PesquisarSQL(String termos) {
72          List<Anuncio_regra> listaAnuncio = new ArrayList<Anuncio_regra>();
73          Pesquisa_dao pesquisa_dao = new Pesquisa_mysql();
74          pesquisa.setTermos(termos);
75
76          long start = new Date().getTime();
77          listaAnuncio = pesquisa_dao.buscarAnunciosPesquisa(termos);
78          long end = new Date().getTime();
79
80          pesquisa.setTempo((int) (end - start));
81          pesquisa.setHits(listaAnuncio.size());
82          return listaAnuncio;
83      }
```

Figura 16 - Método *PesquisarSQL*
Fonte: Autoria Própria

Na Figura 17 exibe-se o método *buscarAnunciosPesquisa* que faz a busca através de uma instrução *SQL*. Utiliza uma conexão com o banco de dados *MySQL*, e para cada retorno, popula-se a lista de Anúncios.

```
    public List<Anuncio_regra> buscarAnunciosPesquisa(String termos) {
27
28          List<Anuncio_regra> listAnuncio = new ArrayList<Anuncio_regra>();
29
30          String sql;
31          sql = ("SELECT ID_ANUNCIO, TITULO, DESCRICAO, DATAHORA FROM ANUNCIO WHERE DESCRICAO LIKE '%" + termos + "%'");
32
33          try {
34              PreparedStatement ps = Factory_mysql.createConnection().prepareStatement(sql);
35              ResultSet rs = ps.executeQuery();
36              while (rs.next()) {
37                  listAnuncio.add(this.popularListaAnuncio(rs));
38              }
39
40          } catch (SQLException e) {
41              System.out.println("ERRO: " + e.getMessage());
42          }
43          return listAnuncio;
44      }
45  }
```

Figura 17 - Método *buscarAnunciosPesquisa*
Fonte: Autoria Própria

Na Figura 18 exibe-se o método *popularListaAnuncio* na classe Pesquisa_*mysql* que insere os anúncios encontrados em uma lista.

```
16      private Anuncio_regra popularListaAnuncio(ResultSet rs) throws SQLException {
17          Anuncio_regra cadAnuncio = new Anuncio_regra();
18          cadAnuncio.setId_anuncio(rs.getInt("ID_ANUNCIO"));
19          cadAnuncio.setTitulo(rs.getString("TITULO"));
20          cadAnuncio.setDescricao(rs.getString("DESCRICAO"));
21          String data = f.format(rs.getTimestamp("DATAHORA"));
22          cadAnuncio.setDatahora(data);
23          return cadAnuncio;
24      }
```

Figura 18 - Método *popularListaAnuncio* da classe Pesquisa_*mysql*
Fonte: Autoria Própria

5.3.3.2 Pesquisar SQL FullText

Na Figura 19 está representado o esquema do método *PesquisarSQLFullText*.

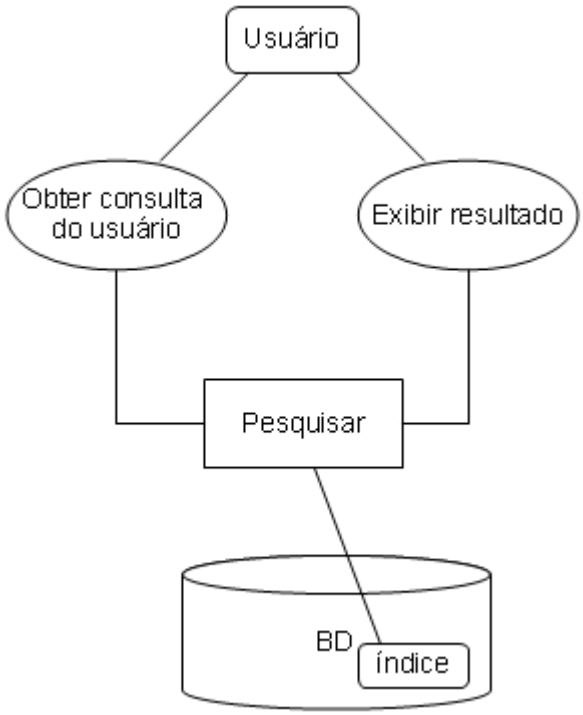


Figura 19 - Diagrama do funcionamento do método *PesquisarSQLFullText*
Fonte: Autoria Própria

Na Figura 20 está o diagrama de sequencia do *Servlet* Pesquisa*SQLFullText*.

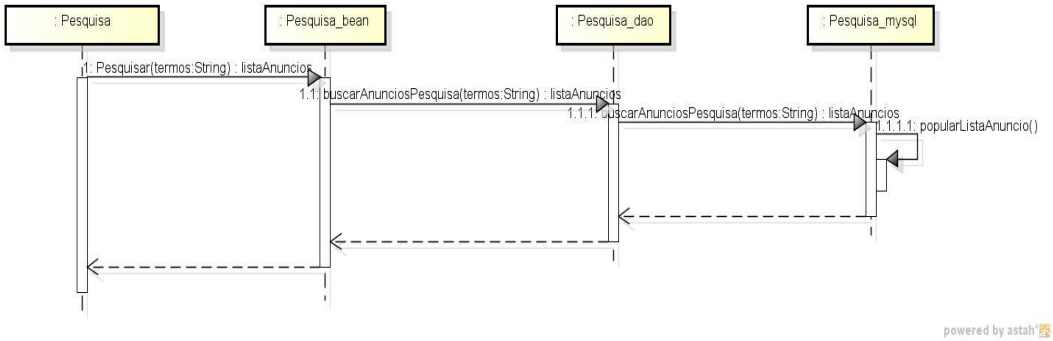


Figura 20 - Diagrama de sequencia do método *PesquisarSQLFullText*
Fonte: Autoria Própria

Na Figura 21 exibe-se o método *PesquisarSQLFullText* que ao fazer a busca passa-se como parâmetro a expressão de busca, e através da instância da classe Pesquisa_dao, retorna-se uma lista já populada na classe Pesquisa_mysql. Setam-se duas variáveis para verificar o tempo de busca em milissegundos.

```
83      public List<Anuncio_regra> PesquisarSQLFullText(String termos) {
84          List<Anuncio_regra> listaAnuncio = new ArrayList<Anuncio_regra>();
85          Pesquisa_dao pesquisa_dao = new Pesquisa_mysql();
86          pesquisa.setTermos(termos);
87
88          long start = new Date().getTime();
89          listaAnuncio = pesquisa_dao.buscarAnunciosPesquisaFT(termos);
90          long end = new Date().getTime();
91
92          pesquisa.setTempo((int) (end - start));
93          pesquisa.setHits(listaAnuncio.size());
94          return listaAnuncio;
95      }
```

Figura 21 - Método *PesquisarSQLFullText*
Fonte: Autoria Própria

Na Figura 22 exibe-se o método *buscarAnunciosPesquisaFT* que faz a busca através de uma instrução *SQL*, buscando no índice da coluna DESCRICAO, da cópia da tabela anuncio, que é a *anuncio_fulltext*. Utiliza uma conexão com o banco de dados *MySQL*, e para cada retorno, popula-se a lista de Anúncios.

```
45
46      public List<Anuncio_regra> buscarAnunciosPesquisaFT(String termos) {
47
48          List<Anuncio_regra> listAnuncio = new ArrayList<Anuncio_regra>();
49
50          String sql;
51          sql = ("SELECT ID_ANUNCIO, TITULO, DESCRICAO, DATAHORA FROM ANUNCIO_FULLTEXT WHERE MATCH (DESCRICAO) AGAINST ('" + termos + "')");
52          try {
53              PreparedStatement ps = Factory_mysql.createConnection().prepareStatement(sql);
54              ResultSet rs = ps.executeQuery();
55              while (rs.next()) {
56                  listAnuncio.add(this.popularListaAnuncio(rs));
57              }
58
59          } catch (SQLException e) {
60              System.out.println("ERRO: " + e.getMessage());
61          }
62          return listAnuncio;
63      }
```

Figura 22 - Método *buscarAnunciosPesquisaFT*
Fonte: Autoria Própria

5.3.3.3 Pesquisar Lucene

Na Figura 23 está representado o esquema do método P*esquisarLucene*.

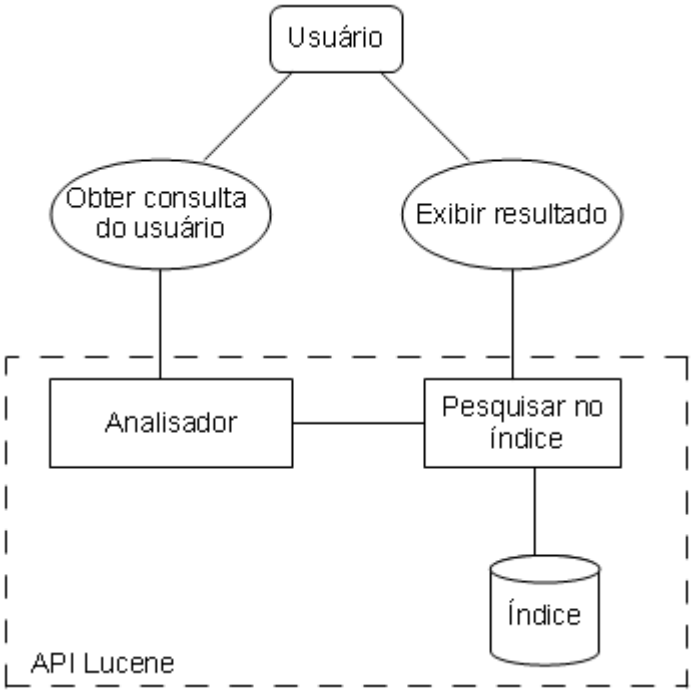


Figura 23 - Diagrama do funcionamento do método *PesquisarLucene*
Fonte: Autoria Própria

Na Figura 24 está o diagrama de sequencia do método *PesquisarLucene*.

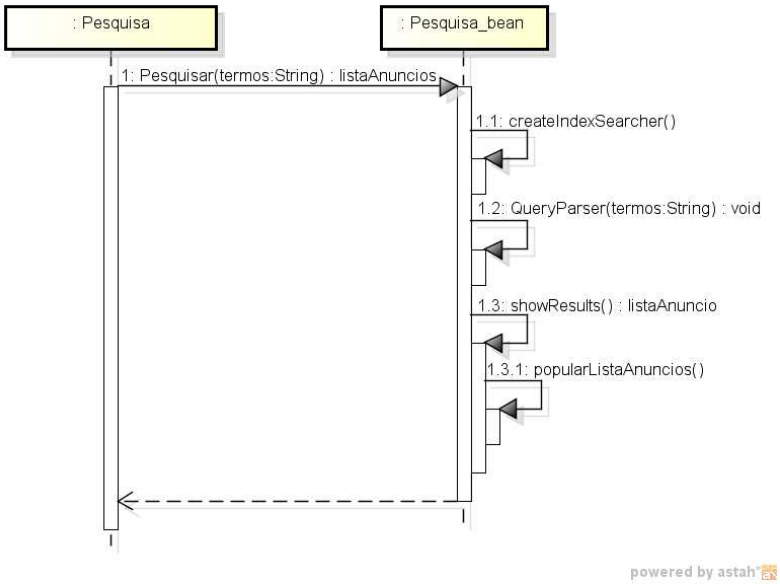


Figura 24 - Diagrama de sequencia do método *PesquisarLucene*
Fonte: Autoria Própria

Na Figura 25 exibe-se o método *createIndexSearch*, onde cria-se uma instância da classe *IndexSearch* com a localização do diretório do índice.

```
30      public void createIndexSearcher() throws CorruptIndexException, IOException {
31          indexSearcher = new IndexSearcher(indexDirectory);
32      }
```

Figura 25 - Método *createIndexSearch*

Fonte: Autoria Própria

Na Figura 26 exibe-se o método *PesquisarLucene* que ao fazer a busca passa-se como parâmetro a expressão de busca. Setam-se duas variáveis para verificar o tempo total de busca em milissegundos. Cria-se uma instância da classe *QueryParser*, setando a busca no campo descrição e utilizando o analisador. Cria-se uma instância da classe *Query*, fazendo o *parser* da busca. O retorno do método é uma lista de anúncios encontrados pelo método *showSearchResults*.

```
61      public List<Anuncio_regra> PesquisarLucene(String termos) throws org.apache.lucene.queryParser.ParseException {
62          pesquisa.setTermos(termos);
63          long start = new Date().getTime();
64          QueryParser queryParser = new QueryParser("descricao", new PortuguesAnalyzer());
65          Query query = queryParser.parse(termos);
66          long end = new Date().getTime();
67          pesquisa.setTempo((int) (end - start));
68          return showSearchResults(query);
69      }
```

Figura 26 - Método *PesquisarLucene*
Fonte: Autoria Própria

Na Figura 27 exibe-se o método *showSearchResults*, onde o objeto *topDocs*, busca como resultado os 20 documentos mais relevantes ao objeto de busca *query*, sendo o *totalHits*, o total de documentos encontrados. Instancia-se um *array ScoreDoc* e para cada documento retornado, a lista de anúncios é populada com o documento. Inicialmente, pensou-se em utilizar *Phrase Query*, *splitando* a *String* em *Terms*. Porém, *PhraseQuery* não utiliza analisador (o mesmo utilizado para indexação). Solução: utilizar *Query Parser*, que utiliza o mesmo analisador e fragmenta em termos a *String* buscada.

```
43      public List<Anuncio_regra> showSearchResults(Query query) {
44          List<Anuncio_regra> listaAnuncio = new ArrayList<Anuncio_regra>();
45          try {
46              TopDocs topDocs = indexSearcher.search(query, 20);
47              System.out.println("Total hits " + topDocs.totalHits);
48              pesquisa.setHits(topDocs.totalHits);
49              ScoreDoc[] scoreDosArray = topDocs.scoreDocs;
50              for (ScoreDoc scoredoc : scoreDosArray) {
51                  Document doc = indexSearcher.doc(scoredoc.doc);
52                  listaAnuncio.add(popularListaAnuncio(doc));
53              }
54          } catch (IOException e) {
55          }
56
57          return listaAnuncio;
58      }
```

Figura 27 - Método *showSearchResults*
Fonte: Autoria Própria

Na Figura 28 exibe-se o método *popularListaAnuncio* que recebe como parâmetro um documento, e popula-se a lista de anúncios com o documento passado por parâmetro.

```
34      private Anuncio_regra popularListaAnuncio(Document doc) {
35          Anuncio_regra cadAnuncio = new Anuncio_regra();
36          cadAnuncio.setId_anuncio(Integer.valueOf(doc.getField("id_anuncio").stringValue()));
37          cadAnuncio.setTitulo(doc.getField("titulo").stringValue());
38          cadAnuncio.setDescricao(doc.getField("descricao").stringValue());
39          cadAnuncio.setDatahora(doc.getField("data").stringValue());
40          return cadAnuncio;
41      }
```

Figura 28 - Método *popularListaAnuncio*
Fonte: Autoria Própria

6 TESTES

Os testes de desempenho validam aplicação utilizando estatísticas apropriadas (Testar.me, 2010). A realização dos testes é imprescindível para validar se a solução proposta neste trabalho atende aos requisitos de agilidade e precisão à resposta de requisições de busca na rede social proposta, realizando um comparativo das tecnologias de busca apresentadas neste trabalho.

Algumas das ferramentas para teste de desempenho *open source* disponíveis no mercado são *Open STA, http_load, The Grinder, Siege, WEBLoad e JMeter* (Library of Wonders, 2009). A ferramenta escolhida para realização dos testes foi a *JMeter* (*Apache Jakarta JMeter*), uma ferramenta *open source* escrita em Java, da organização *Apache Software Foundation*, que pode ser usada para testar o desempenho tanto em recursos estáticos e dinâmicos (arquivos, *Servlets*, *Perl scripts*, Objetos Java, Banco de dados e Consultas, Servidores FTP e mais) (APACHE, 2011). Suportando variáveis aleatórias, teste distribuído (ambiente distribuído), designada para testes funcionais, tempo de carregamento, e estresse, dando *feedback* instantâneo visual e possui comunidade ativa oferecendo suporte a dúvidas de funcionamento da ferramenta.

A ferramenta foi utilizada com o objetivo de analisar graficamente os resultados de diversas requisições simultâneas, porém para qualquer teste que venha a ser feito utilizando o *JMeter*, é necessário criar um Plano de Testes, incluindo os elementos do teste.

Nas seções seguintes são apresentados os testes feitos pelo protótipo, a criação do plano de testes, as configurações da máquina na qual foram realizados os testes, a realização e os resultados dos testes.

6.1 TESTES PELO PROTÓTIPO

Antes de utilizar uma ferramenta para avaliar o desempenho do protótipo, foram realizados testes no protótipo rodando-o no *browser* para verificar se todas as consultas estavam funcionando corretamente. Na Figura 29 é apresentada a tela do protótipo rodando no *browser*.

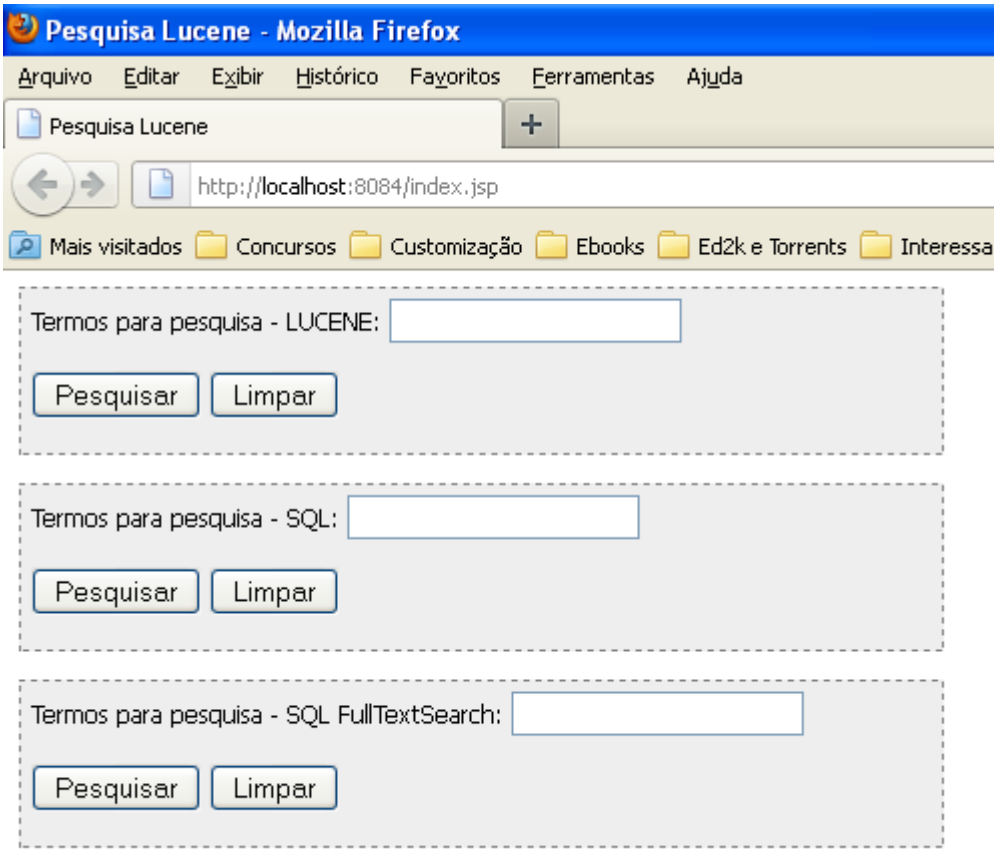


Figura 29 - Tela protótipo
Fonte: Autoria Própria

Ao efetuar a pesquisa sobre o termo república centro, os resultados apresentados são os quais aparecem na Figura 30.

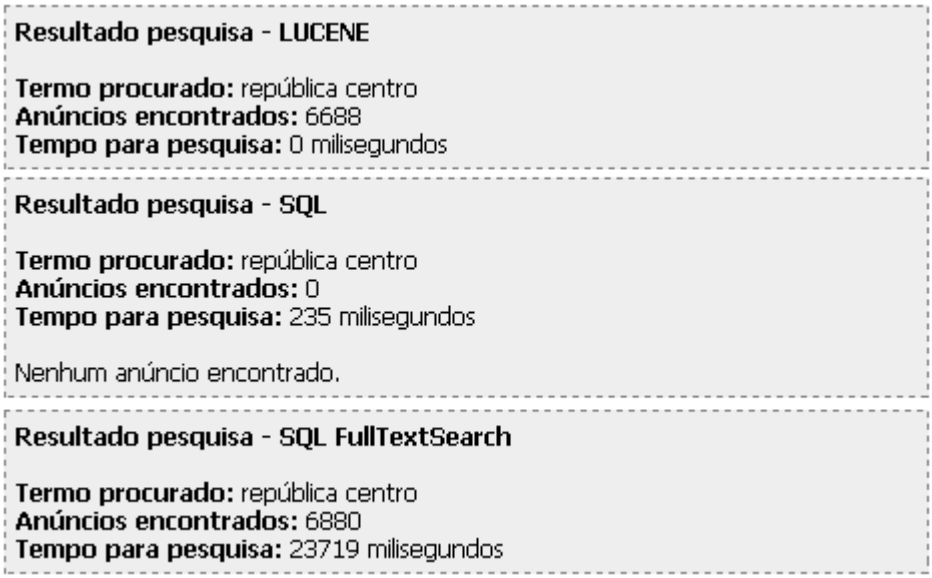


Figura 30 - Resultados protótipo
Fonte: Autoria Própria

Analisando a Figura 30, nota-se que a quantidade de anúncios encontrados pelo motor de busca *SQL* está zerada. Caracterizando um problema com a implementação deste motor de busca. Ao verificar a instrução *SQL* utilizada neste motor os resultados encontrados foram os quais estão apresentados na Figura 31.

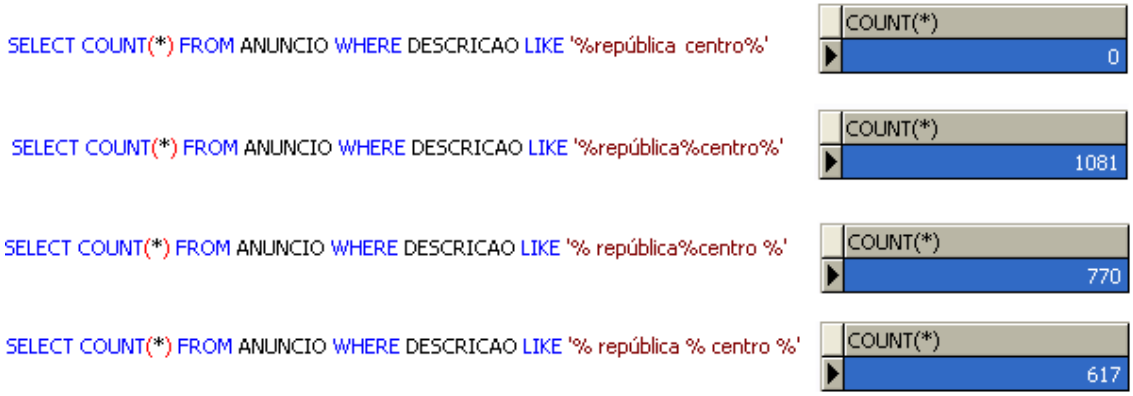


Figura 31 - *SELECT LIKE* no Banco de Dados
Fonte: Autoria Própria

O *SELECT* que foi utilizado nesta consulta foi: *SELECT ID_ANUNCIO, TITULO, DESCRICAO, DATAHORA FROM ANUNCIO WHERE DESCRICAO LIKE '% república centro %'*.

Notou-se que ao inserir um caractere '%' entre os termos buscados ('%república%centro%'), o retorno da instrução *SQL* obteve mais anúncios conforme apresentado na Figura 31. Nesta Figura 31 também é apresentado que o espaço entre os termos buscados influencia no total de anúncios encontrados.

Decidiu-se pela criação de um método que acrescente caracteres "%' sem espaços a cada termo encontrado entre os termos buscados. Mantido este método, novamente realizou-se o teste através do protótipo e o retorno da busca através do motor *SQL* foi o qual é apresentado na Figura 32.

Figura 32 - Resultado *SQL* – correção
Fonte: Autoria Própria

6.2 PLANO DE TESTE

Para a criação do plano de teste foi utilizado o programa *BadBoy* que permite gravar e exportar o plano de teste (*script*) para ser utilizado pelo *JMeter*. O *BadBoy* é

capaz de gravar, como um macro, tudo o que é feito em uma página *Web* como *requests*, parâmetros, *alert*, etc. (NOGUEIRA, 2008). Na Figura 34 apresenta-se a tela inicial do *BadBoy*.

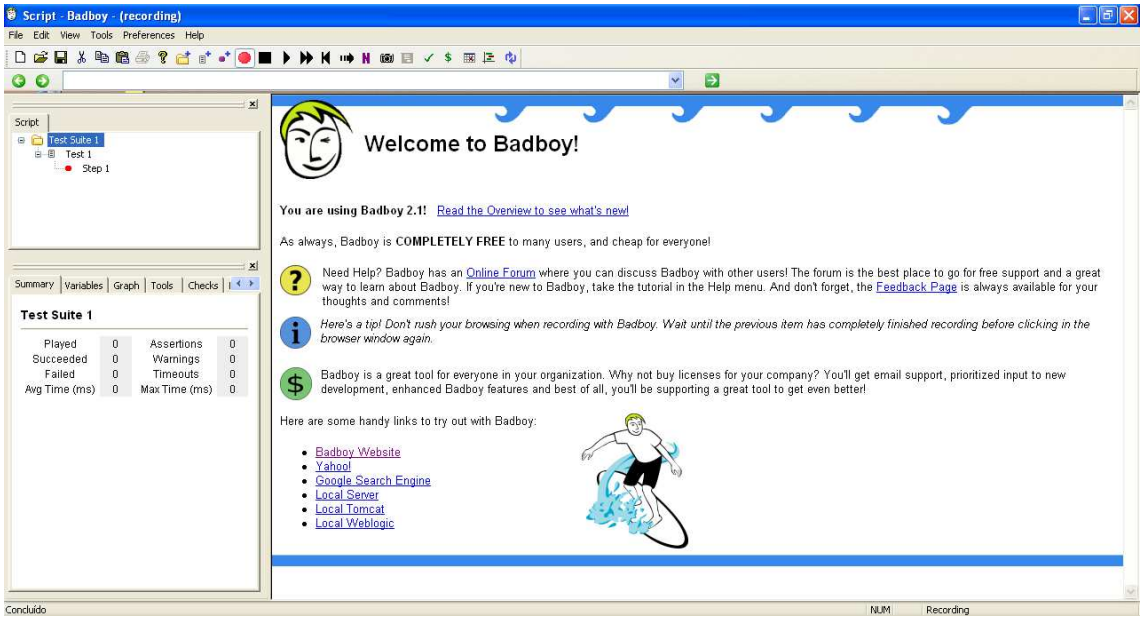


Figura 33 – Tela inicial do programa *BadBoy*
Fonte: Autoria Própria

Foi realizado o acesso ao protótipo na aba de endereço do programa *BadBoy* e simulado a pesquisa de termos para cada um dos três motores de busca, SQL, SQL full-text e Lucene. Estas requisições foram gravadas conforme apresentado na Figura 34.

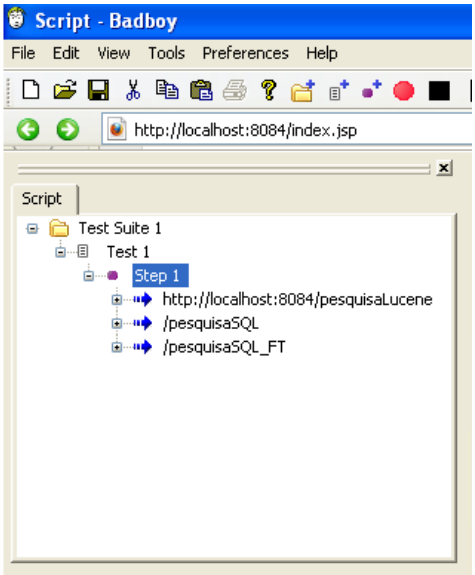


Figura 34 - Plano de teste gravado com a ferramenta *BadBoy*
Fonte: Autoria Própria

Após esta gravação, o plano de testes foi salvo e exportado ao *JMeter*, está apresentado na Figura 35 a abertura do plano de testes no *JMeter*.

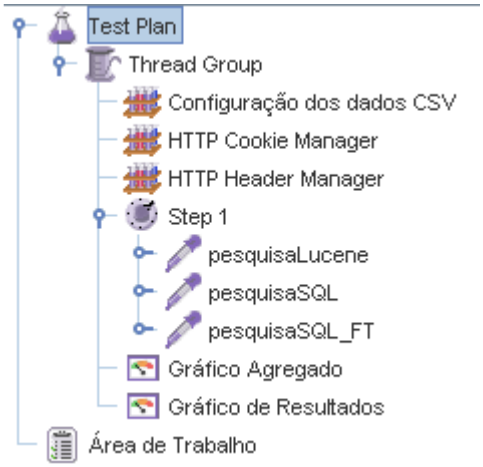


Figura 35 – Plano de teste exportado ao *JMeter*
Fonte: Autoria Própria

O *JMeter* disponibiliza um controle de *threads*, chamado *Thread Group*, no qual é possível configurar o número de *threads*, a quantidade de vezes que cada *thread* será executada e o intervalo entre cada execução, configurações necessárias ao teste de carga em torno do cenário à ser criado, onde o número de *threads* corresponde ao número de usuários requisitando buscas com determinados critérios e obtendo retornos, o intervalo entre cada execução corresponde ao tempo em que todos estes usuários farão suas requisições, sendo este tempo variável em relação aos usuários não fazerem buscas imediatamente ao acessar a rede social. Na Figura 36 apresenta-se o *Thread Group* do *JMeter*.

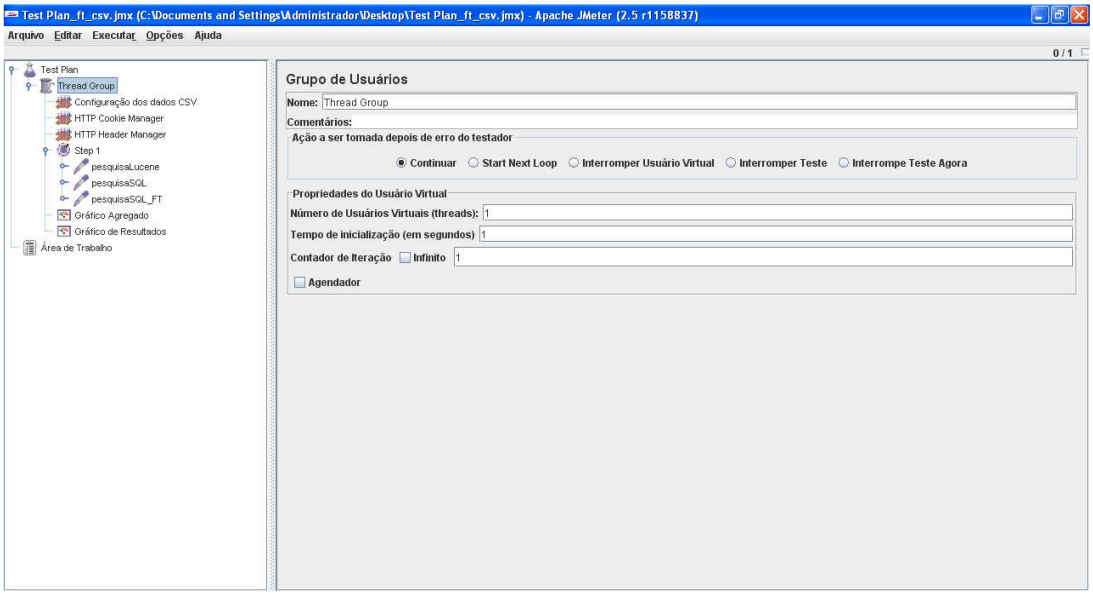


Figura 36 - *Thread Group* do *JMeter*
Fonte: Autoria Própria

Onde os campos da Figura 36 representam:

- **Número de Usuários Virtuais (*threads*):** é a quantidade de *threads* que serão realizadas.
- **Tempo de inicialização (em segundos):** tempo que o *JMeter* leva para instanciar os usuários virtuais.
- **Contador de Iteração:** quantas vezes serão realizados os testes.

O *JMeter* disponibiliza diversos *Listeners*, que são usados para exibir os resultados dos testes em formas visuais de gráficos e tabelas. Na Figura 37 apresenta-se o Gráfico Agregado que foi utilizado para visualização e comparação dos resultados dos testes efetuados.

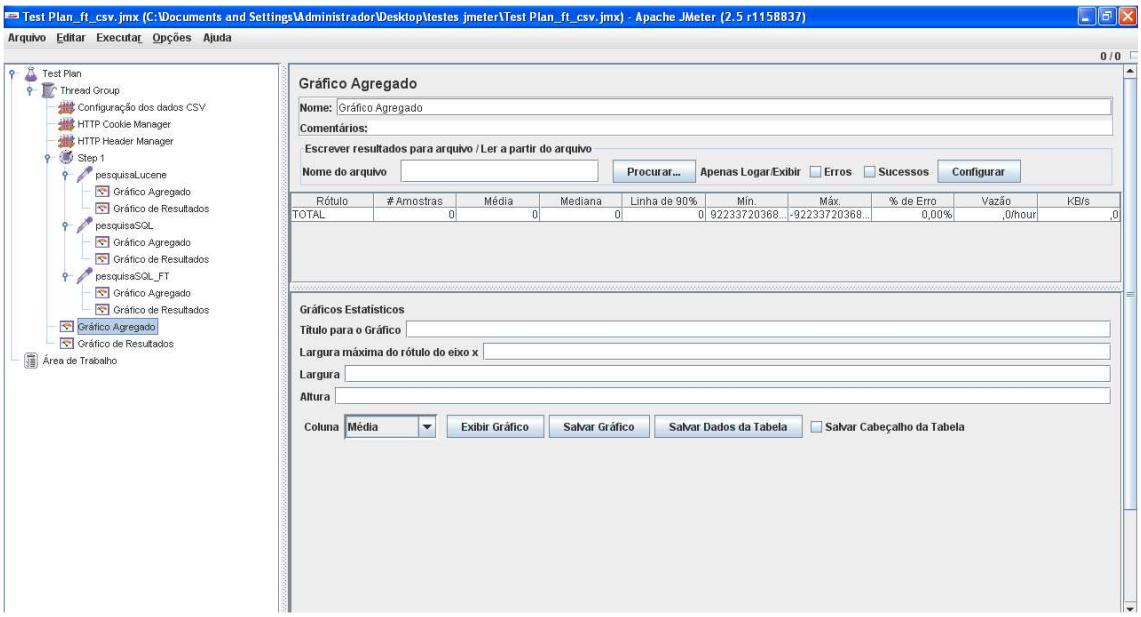


Figura 37 - Gráfico Agregado
Fonte: Autoria Própria

Onde os campos na Figura 37 representam:

- **#Amostras:** é o número de *threads* que foram realizadas.
- **Média:** é o tempo médio de resposta da requisição em milissegundos.
- **Mediana:** é o tempo de resposta utilizando 50% da amostra.
- **Linha de 90%:** Média das amostras utilizando 90% da amostra eliminando 5% no inicial e mais 5% do final da amostra, com o intuito de deixar a amostra mais confiável.

- **Mín.:** é o menor tempo (melhor tempo) de resposta entre todas as requisições em milissegundos.
- **Máx.:** é o maior tempo de resposta entre todas as requisições em milissegundos.
- **% de Erro:** porcentagem de erro entre as requisições.
- **Desvio:** é uma medida da variabilidade de um conjunto de dados. Esta é uma medida padrão estatístico (The Apache Software Foundation, 2011).
- **Vazão:** é o cálculo de pedidos / unidade de tempo. O tempo é calculado a partir do início da primeira amostra até o fim da última amostra. Isso inclui quaisquer intervalos entre as amostras, como é suposto para representar a carga sobre o servidor. A fórmula é: Vazão = (número de pedidos) / (tempo total) (The Apache Software Foundation, 2011).

6.2.1 Máquina Utilizada nos Testes

A máquina onde foram efetuados os testes contém estas configurações de *hardware* e *software*:

Hardware

- Computador Pessoal
- Processador Intel® Core™ i7 950 3.06 GHz
- Memória RAM de 4GB DDR3-1600MHz
- HD Sata II

Software

- Sistema Operacional Microsoft Windows 7 *Ultimate* 64 bits

6.3 REALIZAÇÃO DOS TESTES

Para efetuar os testes com o motor de busca *Lucene*, é necessário efetuar a indexação dos anúncios. O processo de indexação dos 20.000 anúncios levou 5860 milissegundos = 5,86 segundos.

Foi realizado um teste inicial buscando pelo termo “república centro” nos três motores de busca simultaneamente, *SQL*, *SQL full-text* e *Lucene*. Para rodar este teste, é necessário que o protótipo esteja rodando no *browser* e o serviço *MySQL* ativo. O *JMeter* foi configurado conforme a Figura 38.

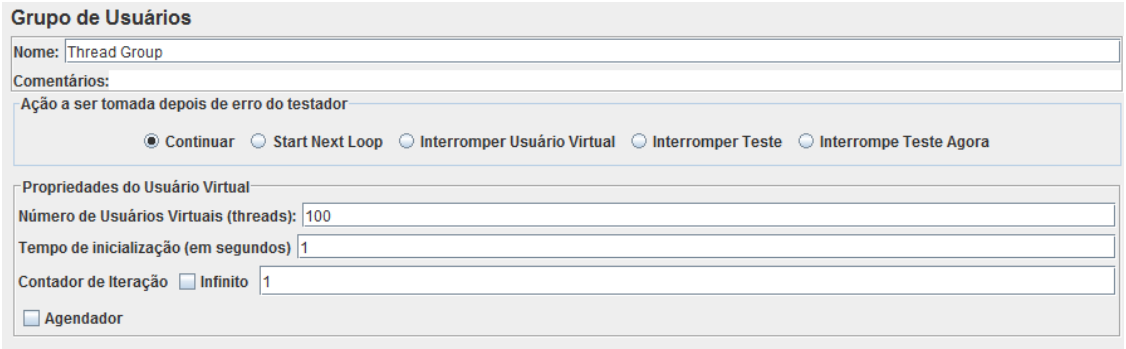


Figura 38 – Configuração – Teste inicial
Fonte: Autoria Própria

Após rodar o teste, o resultado está apresentado na Figura 39.

Rótulo	# Amostras	Média	Mediana	Linha de 90%	Mín.	Máx.	% de Erro	Vazão	KB/s
pesquisaLuce...	98	25248	13150	70558	7	104825	2,04%	55,6/min	11,7
pesquisaSQL	89	49794	45418	99834	601	144130	73,03%	31,3/min	5,0
pesquisaSQL...	88	42615	37960	81062	3248	145195	98,86%	29,7/min	1,0
TOTAL	275	38749	32190	81825	7	145195	56,00%	1,5/sec	12,7

Figura 39 – Resultado – Teste inicial
Fonte: Autoria Própria

O plano de testes travou faltando 12 *threads* para finalizar, buscando pelo termo “república centro” na configuração padrão da máquina utilizada nos testes. Apesar das taxas de erro, o *Lucene* apresentou os melhores resultados de tempo de resposta mínimo e máximo para atender uma requisição e baixa percentagem de erro, demonstrando assim que os outros motores de busca exigem mais hardware para rodarem, pois este teste foi realizado nas configurações padronizadas de alocação de memória das ferramentas utilizadas para execução do teste (*JMeter e NetBeans*).

Foi verificada a causa dos erros da execução deste teste inicial, foram feitos ajustes nas configurações de alocação de memória no *JMeter* e na máquina virtual Java (*NetBeans*). Também o número máximo de conexões múltiplas no *MySQL* foi aumentado:

- Configurações *JMeter*, no arquivo *jmeter.bat*, *HEAP* -Xms512m-Xmx3072m.
- Configurações *NetBeans*: nas propriedades do projeto, opção Executar, Opções da VM, -Xms512m –Xmx2048m. Nas propriedades do servidor *TomCat* na aba plataforma, Opções da VM, -Xms512m –Xmx1024m. No arquivo *netbeans.conf* valores J-Xms512m, J-XX: PermSize512m e –J-XX:MaxPermSize2048m.
- Configurações *MySQL*, no arquivo *my.ini, max_connections* para 10000.

Após estas configurações, os resultados dos testes são apresentados nas seções a seguir.

6.3.1 Pesquisas Com o Termo “república centro”

Nas seções seguintes são apresentados os testes com qual a configuração utilizada e os resultados obtidos com 100, 300 e 500 usuários efetuando a busca com o termo “república centro”.

6.3.1.1 100 usuários – termo

Na Figura 40 são apresentadas as configurações utilizadas neste teste.

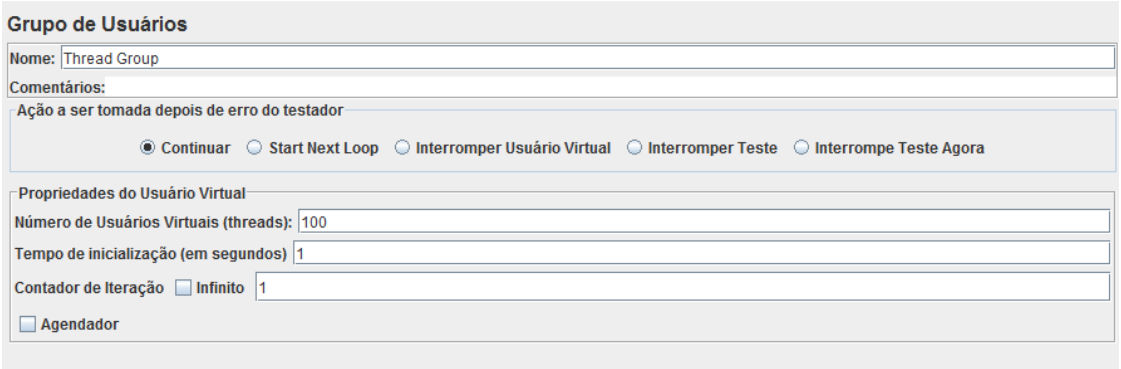


Figura 40 - Configurações 100 usuários - termo
Fonte: Autoria Própria

Na figura 41 são apresentados os resultados obtidos neste teste.

Rótulo	# Amostras	Média	Mediana	Linha de 90%	Mín.	Máx.	% de Erro	Vazão	KB/s
pesquisaLuc...	100	541	284	1098	6	2891	0,00%	25,5/sec	326,9
pesquisaSQL	100	4297	3497	8427	337	11804	0,00%	7,7/sec	234,8
pesquisaSQL...	100	7123	7351	10218	1289	12919	0,00%	5,7/sec	115,9
TOTAL	300	3987	2849	9361	6	12919	0,00%	16,8/sec	355,9

Figura 41 - Resultados 100 usuários – termo
Fonte: Autoria Própria

No Gráfico 2 é apresentado o comparativo de valores mínimo e máximo de resposta às requisições de busca.

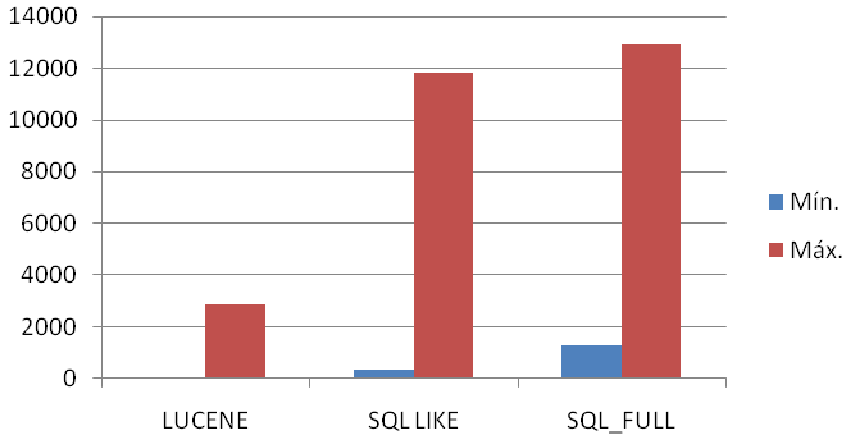


Gráfico 2 - Comparativo mín/máx. 100 usuários - termo
Fonte: Autoria Própria

6.3.1.2 300 usuários – termo

Neste teste, foi necessário aumentar o tempo de inicialização de *threads* para 25 segundos, para que o mesmo não apresentasse taxas de erros. Em virtude de recursos de *hardware* disponíveis e simulando tempo variável de acessos simultâneos à busca.

Na Figura 42 são apresentadas as configurações utilizadas neste teste.

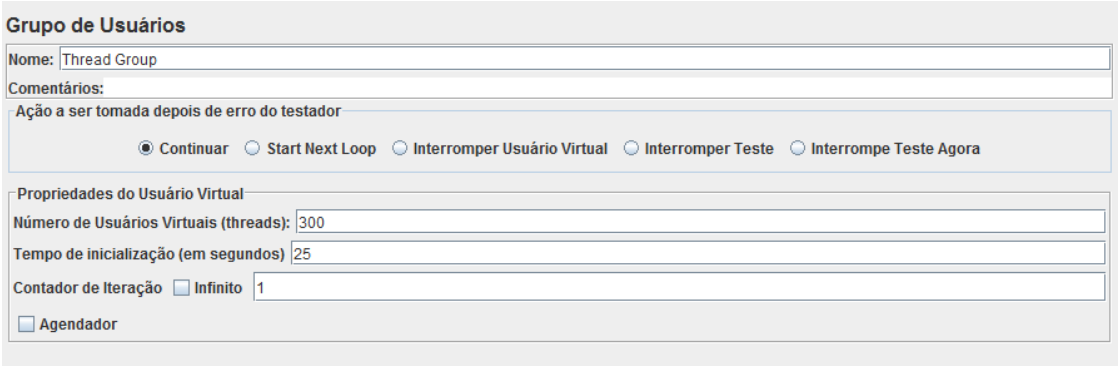


Figura 42 - Configurações 300 usuários – termo
Fonte: Autoria Própria

Na figura 43 são apresentados os resultados obtidos neste teste.

Rótulo	# Amostras	Média	Mediana	Linha de 90%	Mín.	Máx.	% de Erro	Vazão	KB/s
pesquisaLuc...	300	2713	1197	8629	8	15484	0,00%	7,8/sec	100,1
pesquisaSQL	300	5945	5293	12073	734	19800	0,00%	6,2/sec	190,1
pesquisaSQL...	300	8895	7597	16210	1043	21962	0,00%	6,1/sec	123,5
TOTAL	900	5851	4758	13393	8	21962	0,00%	17,7/sec	375,3

Figura 43 - Resultados 300 usuários – termo
Fonte: Autoria Própria

No Gráfico 3 é apresentado o comparativo de valores mínimo e máximo de resposta às requisições de busca.

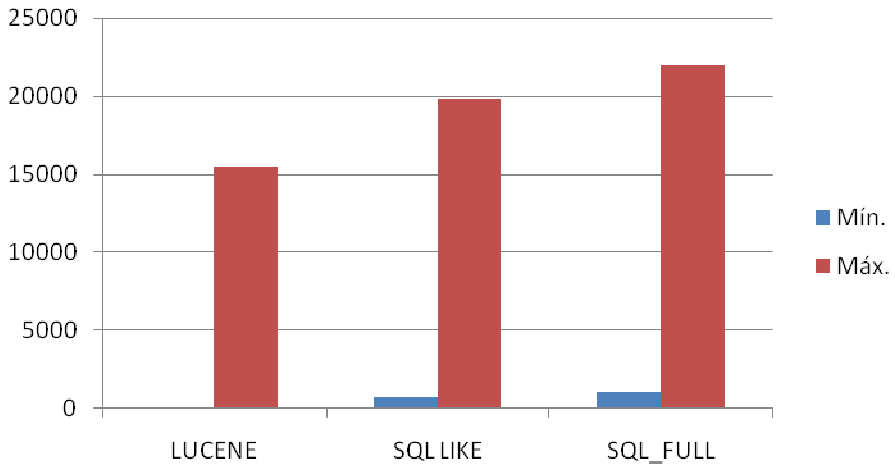


Gráfico 3 - Comparativo mín./máx. 300 usuários - termo
Fonte: Autoria Própria

6.3.1.3 500 usuários - termo

Neste teste, foi necessário aumentar o tempo de inicialização de *threads* para 60 segundos, para que o mesmo não apresentasse taxas de erros. Em virtude de recursos de *hardware* disponíveis e simulando tempo variável de acessos simultâneos à busca.

Na Figura 44 são apresentadas as configurações utilizadas neste teste.

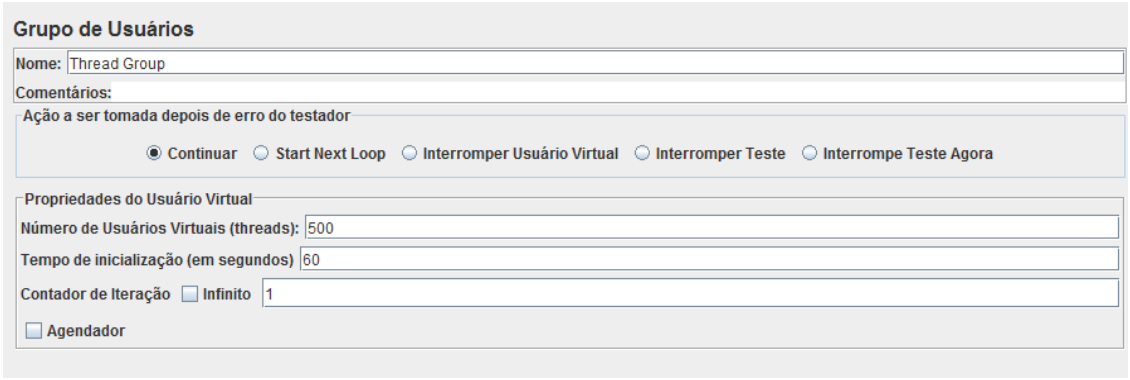


Figura 44 - Configurações 500 usuários – termo
Fonte: Autoria Própria

Na figura 45 são apresentados os resultados obtidos neste teste.

Rótulo	# Amostras	Média	Mediana	Linha de 90%	Mín.	Máx.	% de Erro	Vazão	KB/s
pesquisaLuc...	500	1352	547	4147	5	7694	0,00%	7,5/sec	96,3
pesquisaSQL	500	3521	2785	7171	215	13390	0,00%	6,9/sec	209,8
pesquisaSQL...	500	4906	4715	9461	227	14143	0,00%	6,3/sec	127,6
TOTAL	1500	3260	2240	7897	5	14143	0,00%	18,8/sec	398,7

Figura 45 - Resultados 500 usuários – termo
Fonte: Autoria Própria

No Gráfico 4 é apresentado o comparativo de valores mínimo e máximo de resposta às requisições de busca.

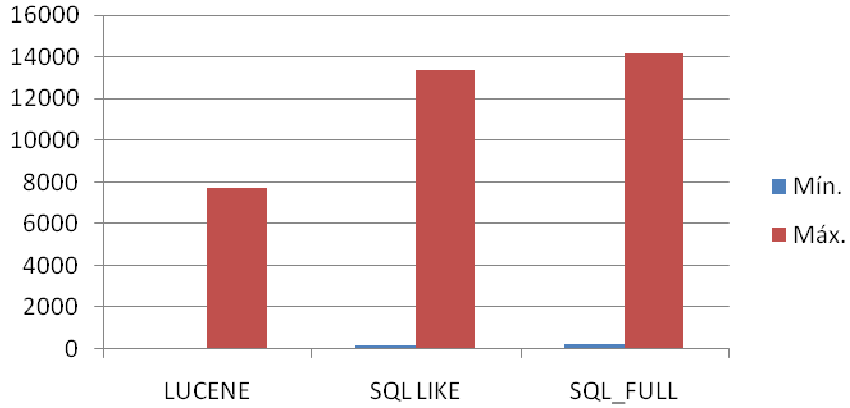


Gráfico 4 - Comparativo mín./máx. 500 usuários - termo
Fonte: Autoria Própria

6.3.2 Pesquisas com lista de termos

Nas seções seguintes são apresentados os testes com qual a configuração utilizada e os resultados obtidos com 100, 300 e 500 usuários efetuando a busca com uma lista de termos:

- república universidade;
- apartamento para dividir;
- vaga em pensionato;
- quarto para alugar;
- dividir quarto;
- colegas para montar república;
- república centro;
- estudantes dividir quarto;
- universitários alugar casa;
- apartamento para universitários;

6.3.2.1 100 usuários – lista de termos

Na Figura 46 são apresentadas as configurações utilizadas neste teste.

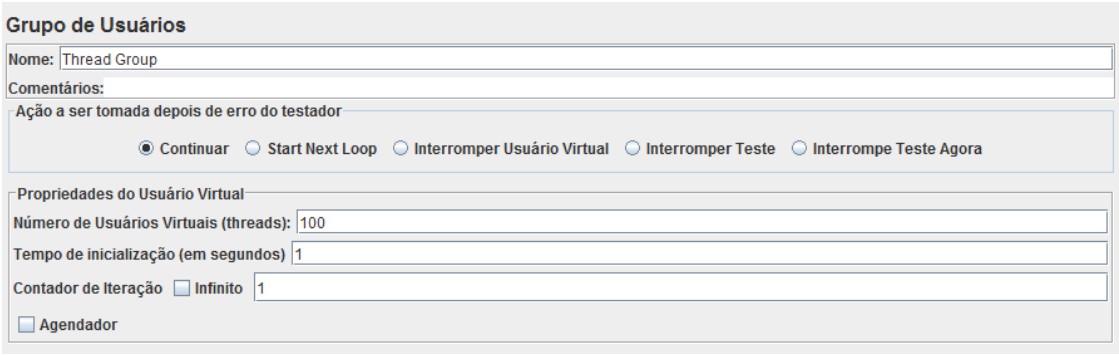


Figura 46 - Configurações 100 usuários - lista de termos
Fonte: Autoria Própria

Na figura 47 são apresentados os resultados obtidos neste teste.

Rótulo	# Amostras	Média	Mediana	Linha de 90%	Mín.	Máx.	% de Erro	Vazão	KB/s
pesquisaLuc...	100	329	212	861	6	1231	0,00%	45,6/sec	837,4
pesquisaSQL	100	4132	3427	8053	228	10367	0,00%	8,7/sec	237,4
pesquisaSQL...	100	5624	5527	9419	534	13365	0,00%	6,2/sec	131,1
TOTAL	300	3362	2416	7989	6	13365	0,00%	18,5/sec	409,5

Figura 47 – Resultados 100 usuários - lista de termos
Fonte: Autoria Própria

No Gráfico 5 é apresentado o comparativo de valores mínimo e máximo de resposta às requisições de busca.

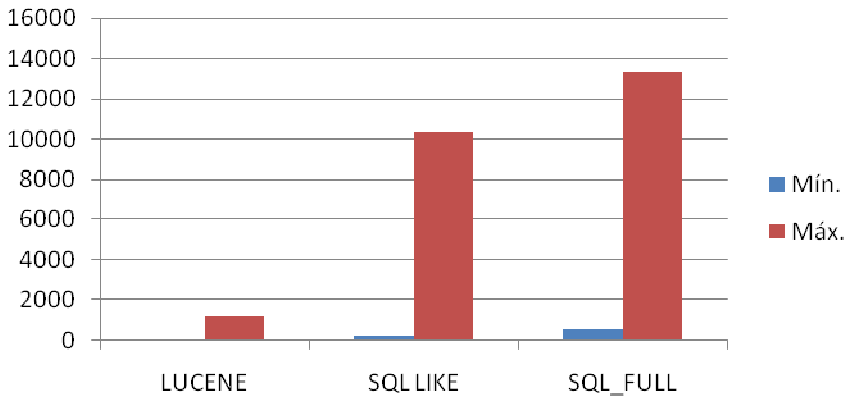


Gráfico 5 - Comparativo mín./máx. 100 usuários - lista de termos
Fonte: Autoria Própria

6.3.2.2 300 usuários – lista de termos

Neste teste, foi necessário aumentar o tempo de inicialização de *threads* para 25 segundos, para que o mesmo não apresentasse taxas de erros. Em virtude de recursos de *hardware* disponíveis e simulando tempo variável de acessos simultâneos à busca.

Na Figura 48 são apresentadas as configurações utilizadas neste teste.

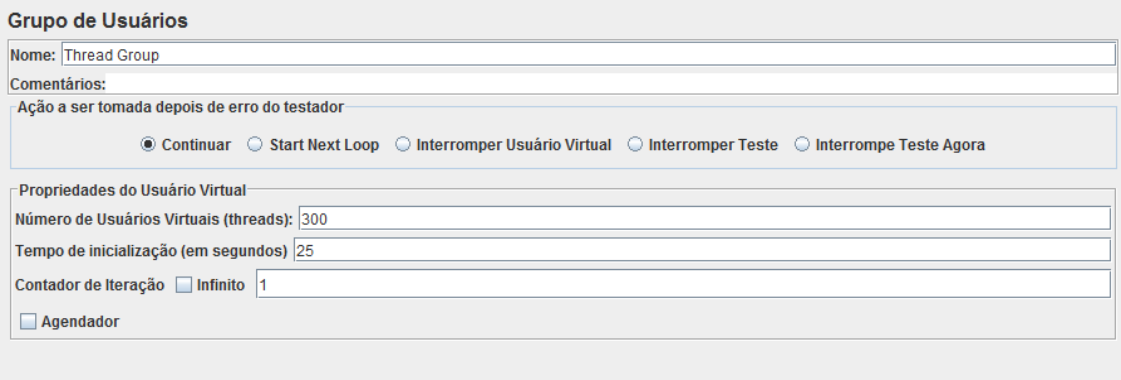


Figura 48 - Configurações 300 usuários - lista de termos
Fonte: Autoria Própria

Na figura 49 são apresentados os resultados obtidos neste teste.

Rótulo	# Amostras	Média	Mediana	Linha de 90%	Mín.	Máx.	% de Erro	Vazão	KB/s
pesquisaLuc...	300	1645	591	3639	5	17251	0,00%	7,7/sec	140,7
pesquisaSQL	300	5021	3407	11123	168	22826	0,00%	5,0/sec	135,0
pesquisaSQL...	300	9524	6600	22721	93	39150	0,00%	4,8/sec	101,0
TOTAL	900	5397	2782	14058	5	39150	0,00%	14,4/sec	319,1

Figura 49 – Resultados 300 usuários - lista de termos
Fonte: Autoria Própria

No Gráfico 6 é apresentado o comparativo de valores mínimo e máximo de resposta às requisições de busca.

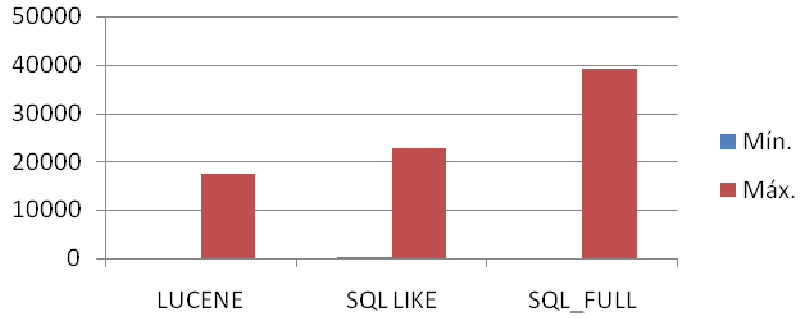


Gráfico 6 - Comparativo mín./máx. 300 usuários - lista de termos
Fonte: Autoria Própria

6.3.2.3 500 usuários – lista de termos

Neste teste, foi necessário aumentar o tempo de inicialização de *threads* para 60 segundos, para que o mesmo não apresentasse taxas de erros. Em virtude de recursos de *hardware* disponíveis e simulando tempo variável de acessos simultâneos à busca.

Na Figura 50 são apresentadas as configurações utilizadas neste teste.

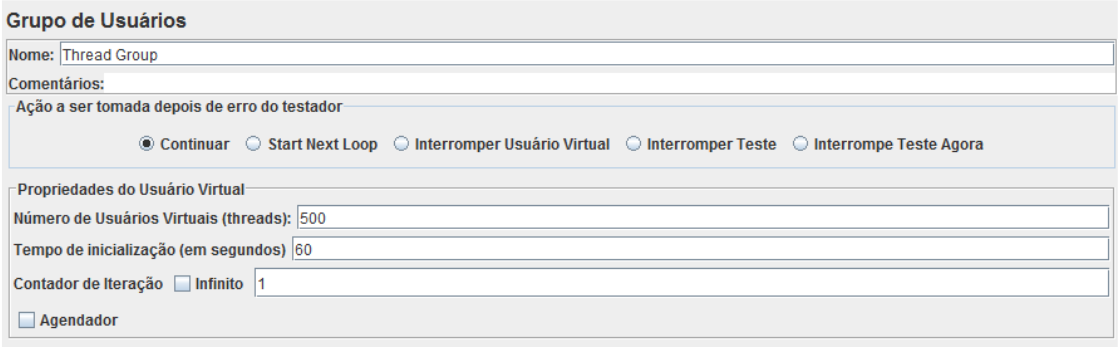


Figura 50 - Configurações 500 usuários - lista de termos
Fonte: Autoria Própria

Na figura 51 são apresentados os resultados obtidos neste teste.

Rótulo	# Amostras	Média	Mediana	Linha de 90%	Mín.	Máx.	% de Erro	Vazão	KB/s
pesquisaLuc...	500	726	344	1916	5	4864	0,00%	7,8/sec	144,0
pesquisaSQL	500	2320	1751	4766	171	12779	0,00%	7,2/sec	194,7
pesquisaSQL...	500	3339	2765	7227	65	14313	0,00%	6,9/sec	144,7
TOTAL	1500	2128	1418	5052	5	14313	0,00%	20,6/sec	457,4

Figura 51 - Resultados 500 usuários - lista de termos
Fonte: Autoria Própria

No Gráfico 7 é apresentado o comparativo de valores mínimo e máximo de resposta às requisições de busca.

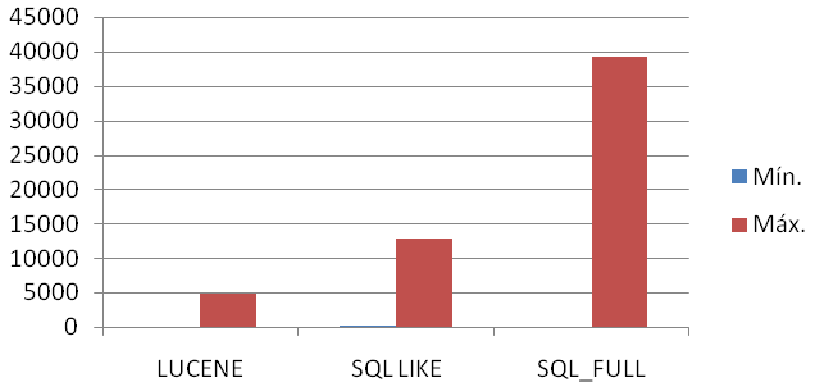


Gráfico 7 - Comparativo mín./máx. 500 usuários - lista de termos
Fonte: Autoria Própria

6.4 RESULTADOS DOS TESTES

Através dos resultados dos testes, os gráficos demonstraram que o *Lucene* obteve melhores desempenhos em tempo mínimo, médio e máximo de resposta para cada requisição solicitando busca, comprovando ser uma biblioteca que oferece agilidade no tempo de resposta e confiabilidade nas respostas das requisições. Porém os piores resultados ficaram com a proposta de *Full-text search*,

que apresentou resultados superiores ao esperado. Provavelmente este desempenho deu-se em virtude da base de dados testada conter pouco volume de informação sendo 20.000 anúncios compreendendo título e descrição variando de 20 a 250 palavras.

Avaliando resultados que apresentaram erros, a implementação *SQL* necessita em torno de 12% e a implementação *SQL Full-text search* necessita de 15% a mais de recursos de *hardware* em relação ao *Lucene*.

O *Lucene* oferece flexibilidade na busca e nos resultados, indo contra as desvantagens apresentadas para *SQL like* e *full-text search* que não oferecem customização, é inviável e sua migração para outras plataformas é dispendiosa.

Foram feitos testes no protótipo para avaliar a relevância dos anúncios encontrados, lembrando que a busca nos três casos, é realizada na coluna DESCRICAO, e de acordo com os resultados, a relevância é diferente para cada motor de busca. Visto que, o motor *Lucene* realiza a relevância de acordo com a frequência do termo em que aparece no documento e atribui um *score* (pontuação); o motor *SQL* realiza uma varredura na coluna satisfazendo a condição descrita no operador *LIKE* e o motor *SQL Full-text search* efetua um cálculo de pontuação referente à quantidade de vezes em que esta apareça na coluna, quanto mais rara mais alta sua pontuação e quanto mais repetida, mais baixa será sua pontuação. Notou-se o seguinte comportamento ao executar o teste com o termo “república centro”:

- *SQL* – nos resultados exibidos, a relevância deu-se em que todos os anúncios retornados continham as palavras que compõem o termo pesquisado, indiferente de quantas palavras havia entre eles.
- *SQL full-text search* – nos resultados exibidos, a relevância deu-se em notar que as palavras que compõem o termo pesquisa, possuem baixa pontuação, (são frequentes nos anúncios) tendo menos chance de exibir resultados relevantes, sendo mais provável exibir resultados irrelevantes.
- *Lucene* – nos primeiros resultados exibidos, a relevância deu-se contendo as duas palavras que compõem o termo pesquisado, sendo os últimos resultando contendo apenas uma ou as duas palavras do termo.

7 CONCLUSÕES

Este trabalho descreveu a proposta de uma rede social para centralização de informações necessárias ao indivíduo que está fora de sua cidade natal. Foram levantados os requisitos iniciais, definido grupo de usuários em potencial (para estimar volume de dados e requisições), apresentados, implementados e submetidos a testes de desempenho, três motores de busca textuais (*SQL*, *SQL full-text* e *Lucene*) como solução à busca ágil e precisa.

Nos testes de desempenho, a biblioteca de pesquisa *Lucene* ofereceu resultados superiores ao esperado e é inovadora em comparação com os demais motores de buscas abordados, oferecendo flexibilidade e expansão de recursos.

Com o motor de busca definido (*Lucene*) e futuramente implementado na rede social proposta, os indivíduos poderão encontrar seu “nicho”, participar e colaborar com os demais usuários que buscam informações relacionadas à moradia, transporte, serviços emergenciais e de entretenimento nas cidades onde está a universidade ou local de trabalho escolhido. E se, o objetivo de não se delimitar às barreiras geográficas e culturas prevalecer na implementação da rede social, é necessária a realização de maior carga de testes, utilizando recursos de software e hardware compatíveis com este cenário pretendido e sem contar nos recursos de idiomas, tradutores e demais requisitos que demandem da expansão da rede social proposta neste trabalho.

Nas seções seguintes são apresentadas as principais dificuldades encontradas e proposta para trabalhos futuros.

7.1 PRINCIPAIS DIFICULDADES ENFRENTADAS

As principais dificuldades enfrentadas foram:

- A criação de uma lista de palavras de parada (*stopwords*) em português, para comparação com as palavras removidas na implementação da classe *Analyzer* do *Lucene*, na tabela que implementa o índice *full-text search*, pois não foi possível a atribuição de privilégios para a alteração destas configurações que são restritas ao administrador do banco de dados;
- A configuração de alocação de memória das ferramentas utilizadas no desenvolvimento do protótipo e necessárias para o desempenho do teste ser satisfatório, visando 0% de erros;

- Materiais (artigos, livros) tratando do desempenho do operador *LIKE* em outros SGBDs.

7.2 TRABALHOS FUTUROS

Visando trabalhos futuros a serem desenvolvidos a partir deste, espera-se que ocorra o desenvolvimento dos requisitos que não foram desenvolvidos neste protótipo, a implementação e publicação desta rede social na Internet e demais possibilidades, tais como:

- Busca por localização geográfica, através das coordenadas de latitude e longitude do endereço da instituição de ensino pretendida ou outro endereço à escolha do usuário como ponto de referência, traçando um raio em torno do ponto de referência, abrangendo anúncios próximos ao local escolhido.
- Estudar motor de pesquisa e indexação do *Lucene* em tempo real visando soluções na concorrência de acesso ao índice.
- Ao efetuar operações de atualização e exclusão dos documentos indexados, estudar rotinas de atualização do índice.
- Estudar formas de otimizar o desempenho do índice do *Lucene*
- Expandir o analisador do *Lucene* com funções de correção ortográfica, sinônimos, *stremming* (redução das palavras à raiz, sendo a raiz um conjunto de caracteres que está presente em todas as derivações desta) (RAPPA, 2010).
- Explorar recursos de busca do *Lucene* utilizando outras implementações da classe *Query*, conforme citadas neste trabalho.
- Expandir a busca do *Lucene* com filtros adicionais, facetamento e *higlighting* (realce dos termos buscados nos resultados da busca).
- Estruturação de dados para organizar conteúdo em diversos formatos, e possibilidade de maior utilização de funcionalidades de indexação e ranking de resultados do *Lucene*. (TEIXEIRA e DUQUE, 2010)
- Conhecer a ferramenta *Solr* que é *open source* e uma plataforma baseada no projeto *Apache Lucene* que possui interface de administração em HTTP e suporta saídas em XML e JSON.

- Conhecer a ferramenta *ElasticSearch* que é um motor de busca baseado em *Lucene*, desenvolvido para ser escalável, distribuído e com o modelo de dados baseado em JSON.

REFERÊNCIAS

AGUIAR, S. **Redes sociais na Internet – Um desafio à pesquisa**. In: XXX Congresso de Brasileiro de Ciências da Comunicação. Santos. 2007. Disponível em: <http://www.sitedaescola.com/downloads/portal_aluno/Maio/Redes%20sociais%20na%20internet-%20desafios%20%E0%20pesquisa.pdf>. Acesso em: 19 mai. 2011.

ALBUQUERQUE, A.; MESQUITA, D.; COSTA, L. **FOLKSONOMIA: uma nova modalidade de indexação e recuperação da informação na Web**. In: Encontro Nacional de Estudantes de Biblioteconomia, Documentação, Gestão, e Ciência da Informação. Os desafios do profissional da informação frente às tecnologias e suportes informacionais do século XXI: lugares de memória para a biblioteconomia. Paraíba. 2010. Disponível em: <http://dci.ccsa.ufpb.br/enebd/index.php/enebd/article/viewFile/183/178>. Acesso em: 18 mai. 2011.

Apache Lucene Eurocon Barcelona. Europe's Premier Lucene and Solr User Conferecence. 19-20 October, Barcelona, Espanha 2011. Disponível em <http://2011.lucene-eurocon.org/pages/120874>. Acesso em: 06 ago. 2011.

AZEVEDO, L. **Redes Sociais Virtuais**. 7 de abr. 2009. Disponível em: <http://blog.censanet.com.br/2009/04/redes-sociais-virtuais/>. Acesso em: 14 mar. 2011

BARROS, L. A. **Suporte a Ambientes Distribuídos para Aprendizagem Cooperativa**. Tese de Doutorado. COPPE/UFRJ, RJ, Brasil. 1994.

BELL, G. **Criando Aplicações para redes sociais**. Tradução Thaís Cristina Casson. São Paulo. Novatec Editora. Sebastopol, Calif. O'Reilly, 2010.

BIANCHI, W. MySQL FullText Search**. SQL Magazine 67**. 20 ago. 2009. Disponível em: <http://www.devmedia.com.br/articles/viewcomp.asp?comp=14040>. Acesso em: 02 ago. 2011.

BLATTMANN, U.; SILVA, F. C. C. da. Colaboração e interação na Web 2.0 e biblioteca 2.0. **Revista ACB: Biblioteconomia em Santa Catarina**, Florianópolis, v.12, n.2, p. 191-215, jul./dez., 2007. Disponível em: <http://www.acbsc.org.br/revista/ojs/viewarticle.php?id=242>. Acesso em: 15 mai. 2011.

BOLETINES PANDALABS. **Redes Sociales en el punto de mira**. Panda Security. 2008. Disponível em: <http://www.pandasecurity.com/img/enc/Red_Soc_punto_mira.pdf>. Acesso em: 25 mai. 2011.

BOYD, D. M.; ELLISON, N. B. Social network sites: Definition, history, and scholarship. **Journal of Computer-Mediated Communication**, 13, 2007. Disponível em: < http://jcmc.indiana.edu/vol13/issue1/boyd.ellison.html>. Acesso em: 10 jun. 2011.

BURKE, P. **Uma história social do conhecimento: de Gutenberg a Diderot**. Rio de Janeiro: Jorge Zahar, 2003. Disponível em: <http://www.scielo.br/pdf/ha/v10n22/22707.pdf>. Acesso em: 13 mai. 2011.

CAELUM. **Java para Desenvolvimento Web**. Disponível em: < http://www.caelum.com.br/download/caelum-java-Web-fj21.pdf>. Acesso em 30 de nov. 2010.

FERREIRA, L. **Analfabetos são mais que o dobro dos universitários**. R7 Notícias. 08 set. 2010. Disponível em: <http://noticias.r7.com/vestibular-e-concursos/noticias/analfabetos-sao-mais-que-o-dobro-dos-universitarios-20110907.html>. Acesso em: 16 jul. 2011.

FERREIRA, T. **Otimização de Consultas MySQL - Parte 01**. 22 jan. 2010. Disponível em: <http://www.forumWeb.com.br/artigo/81/mysql/otimizacao-de-consultas-mysql---parte-01>. Acesso em: 20 ago. 2011.

FERREIRA, T. **Otimização de Consultas MySQL - Parte 02**. 22 jan. 2010. Disponível em: <http://www.forumWeb.com.br/artigo/82/mysql/otimizacao-de-consultas-mysql---parte-02>. Acesso em: 20 ago. 2011

GOSPODNETIC, O.; HATCHER, E.; MCCANDLESS, M. **Lucene in Action**. Second Edition. Stamford, CT. 2010.

GUEDES, R. de M.; DIAS, E. J. W. Indexação Social: abordagem conceitual. **Revista ACB: Biblioteconomia em Santa Catarina**, Florianópolis, v.15, n.1, p. 39-53, jan./jun. 2010. Disponível em: <http://revista.acbsc.org.br/index.php/racb/article/view/686/pdf_17>. Acesso em: 25 mai. 2011.

GUINCHAT, C.; MENOU, M. **Introdução geral às ciências técnicas da informação e documentação**. Brasília: IBICT, 1994. 540 p.

HJORLAND, B. Toward a theory of aboutness, subject, topicality, theme, domain, field, content... and relevance. **Journal of the American Society for Information Science and Technology**, 52, p. 249-298, 2001.Disponível em: < http://pt.scribd.com/doc/55317200/Hj%C3%B8rland-Birger-Towards-a-Theory-of-Aboutness-Subject-Topicality-Theme-Domain-Field-Content-and-Relevance>. Acesso em: 27 mai. 2011.

HJORT, R. Garantindo desempenho com o operador LIKE. **SQL Magazine 52**. 04 mar. 2008. Disponível em: <http://www.devmedia.com.br/articles/viewcomp.asp?comp=8501#>. Acesso em: 05 set. 2011.

IBGE, **Censo Demográfico 2000**. 2003. Disponível em: <http://www.ibge.gov.br/home/presidencia/noticias/27062003censo.shtm>. 2003. Acesso em: 09 set. 2011.

JAVA (LINGUAGEM DE PROGRAMAÇÃO). In: **WIKIPÉDIA**, a enciclopédia livre. Flórida: Wikimedia Foundation, 2011. Disponível em: <http://pt.wikipedia.org/w/index.php?title=Java_(linguagem_de_programa%C3%A7%C3%A3o)&oldid=27103529>. Acesso em: 6 out. 2011.

Library of Wonders. **JMeter 101: Introduction**. 18 de mar. 2009. Disponível em: <http://girliemangalo.wordpress.com/2009/03/18/jmeter-101_introduction/>. Acesso em: 25 set. 2011.

LANCASTER, F. W. **Indexação e resumos: teoria e prática**. Brasília: Briquet de Lemos, 2004. 452 p.

LE MONDE. Infográfico. **Réseaux sociaux: des audiences différentes selon les continents**. 14 de jan. 2008. Disponível em: < http://www.lemonde.fr/technologies/infographie/2008/01/14/reseaux-sociaux-des-audiences-differentes-selon-les-continents_999097_651865.html>. Acesso em: 23 mar. 2011.

LÉVY, P. **Cyberdemocratie**. Paris: Odile Jacob, 2002.

Lucene & Solr User Conference. Conference e-Guide. Lucene Revolution. May 25-26 San Francisco, Ca 2011. Disponível em: <http://lucenerevolution.com/sites/default/files/Eguide_LuceneRevolution_2011_v1d.pdf>. Acesso em: 06 ago. 2011.

LUVIZOTTO, C.; VIDOTTI, S.. Redes Sociais e Comunidades Virtuais Para a Preservação e Transmissão das Tradições Gaúchas na Internet. **Inf. & Soc.: Est.**, João Pessoa, v.20, n.2, p. 77-88, maio/ago. 2010. Disponível em: <http://www.ies.ufpb.br/ojs2/index.php/ies/article/view/6962/4359>. Acesso em: 17 nov. 2011.

Manual de Referência do MySQL 4.1**. Pesquisa Full-text no MySQL**. Disponível em: <http://dev.mysql.com/doc/refman/4.1/pt/fulltext-search.html>. Acesso em: 07 set. 2011.

MARTELETO, R. M. Análise de Redes Sociais – aplicação nos estudos de transferência da informação. **Ci. Inf.,** Brasília, v. 30, n. 1, p. 71-81, jan./abr. 2001. Disponível em: <http://www.scielo.br/pdf/ci/v30n1/a09v30n1.pdf>. Acesso em: 14 jun. 2011.

MARTÍNEZ, S. L. **Internet como medio y objeto de estudio en antropología. Antropologia e Internet**, ago. 2000. Disponível em: <http://www.plazamayor.net/antropologia/archtm/internet/index.html>. Acesso em: 15 jul. 2011.

MATURANA, H. R. **A Ontologia da Realidade**. Editora UFMG, Belo Horizonte, 1997.

MOLINA, J. L.; AGUILAR, C. **Redes sociales y antropología: un estudio de caso (redes personales y discursos étnicos entre jóvenes en Sarajevo)**. Disponível

em: <http://revista-redes.rediris.es/recerca/jlm/public_archivos/Redyant.pdf>. Acesso em: 25 ago. 2011.

MOURA, M. A. **Informação, Ferramentas Ontológicas e Redes Sociais: A interoperabilidade de construção de tesauros e ontologias**. In: IX CONGRESO ISKO CAPÍTULO ESPAÑOL, 2009, Valenza. Anais do IX congreso ISKo capítulo Español. Valenza: Editorial de la UPV, 2009. v. 1. p. 589-604. Disponível em: <http://dci2.ccsa.ufpb.br:8080/jspui/bitstream/123456789/628/1/2396-5184-1-PB.pdf>. Acesso em: 10 abr. 2011.

MSDN. **Full-Text Search (SQL Server).** Disponível em: <http://msdn.microsoft.com/en-us/library/ms142571.aspx>. Acesso em: 27 jul. 2011.

NetBeans. **NetBeans IDE 7.0 Features**. 2011. Disponível em: <http://netbeans.org/features/index.html>. Acesso em: 20 ago. 2011.

NIELSEN, J. **Participation Inequality: Encouraging More Users to Contribute**, 2006. Disponível em: <http://www.useit.com/alertbox/participation_inequality.html>. Acesso em: 18 ago. 2011.

NOGUEIRA, E. **Conhecendo o BadBoy - parte 1**. Sem Bugs. 20 jan. 2008 Disponível em: <http://sembugs.blogspot.com/2008/01/conhecendo-o-bdboy-parte-1.html>. Acesso em: 18 set. 2011.

O'REILLY, T. **What Is Web 2.0 - Design Patterns and Business Models for the Next Generation of Software**. O'Reilly Publishing, 30 set. 2005. Disponível em: <http://oreilly.com/Web2/archive/what-is-Web-20.html>. Acesso em: 12 abr. 2011.

PostgreSQL. **PostgreSQL 8.3.16 Documentation**. Full Text Search. Disponível em: <http://www.postgresql.org/docs/8.3/static/textsearch.html>. Acesso em: 14 set. 2011.

PRIMO, A. **Interação mediada por computador: comunicação, cibercultura, cognição**. Porto Alegre: Sulina, 2007.

RAFFERTY, P.; HIDDERLEY, R. Flickr and democratic indexing: dialogic approaches to indexing. **Aslib Proceedings***,* v. 59, Issue 4/5, 2007. p. 397-410. Disponível em: <http://www.emeraldinsight.com/Insight/viewPDF.jsp?Filename=html/Output/Published/EmeraldFullTextArticle/Pdf/2760590407.pdf>. Acesso em: 15 abr. 2011.

RAPPA, C. **Lucene: uma biblioteca Open Source de indexação e consulta**. O Globo Blogs. 2011. Disponível em: <http://oglobo.globo.com/blogs/tecnologia/posts/2011/03/14/lucene-uma-biblioteca-open-source-de-indexacao-consulta-362382.asp>. Acesso em: 01 set. 2011.

Redação Oficina da Net. **Conheça um pouco sobre o MySQL**. 2 ago. 2007. Disponível em: <http://www.oficinadanet.com.br/artigo/390/conheca_um_pouco_sobre_o_mysql>. Acesso em: 18 ago. 2011.

Redação Oficina da Net. **MySQL - o que é?** 6 jan. 2010. Disponível em: <http://www.oficinadanet.com.br/artigo/2227/mysql_-_o_que_e>. Acesso em: 18 ago. 2011.

ROCHA, C. M. F. **As redes em saúde: entre limites e possibilidades**, 2005. Disponível em: <http://www.opas.org.br/rh/admin/documentos/Estar_em_rede.pdf>. Acesso em: 19 jul. 2011.

Rodrigues, J. Lucene: programar um motor de busca. **Programar - A revista portuguesa de programação**. Revista n° 15. Agosto 2008. Disponível em: <http://www.slideshare.net/filipebes/revista-programar-15>. Acesso em: 19 set. 2011.

ROWLEY, J. **A biblioteca eletrônica**. Brasília: Briquet de Lemos, 2002. 399 p.

SOFTWARE COLABORATIVO. In: **WIKIPÉDIA**, a enciclopédia livre. Flórida: Wikimedia Foundation, 2011. Disponível em: <http://pt.wikipedia.org/w/index.php?title=Software_colaborativo&oldid=26825514>. Acesso em: 12 out. 2011.

Softech Network. **Curso Java para Web**. 2006. Disponível em: <http://www.softechnetwork.com.br/java/CursoJavaWeb.pdf>. Acesso em: 25 set. 2011.

SONAWANE, A. Código fonte indexação. Disponível em <http://www.ibm.com/developerworks/br/java/library/os-apache-lucenesearch/#download>. Acesso em: 25 set. 2011.

_____. Usando o Apache Lucene para Procura de Texto. DeveloperWorksBr - IBM. 2009. Disponível em <http://www.ibm.com/developerworks/br/java/library/os-apache-lucenesearch/>. Acesso em: 25 set. 2011.

SPUDEIT, D. F. A. O. O fenômeno social das redes de informação: reflexão teórica. The phenomenon of social networks information: theoretical reflection. **Revista ACB,** América do Norte, 15, dez. 2009. Disponível em: <http://revista.acbsc.org.br/index.php/racb/article/view/709>. Acesso em: 14 jul. 2011.

TAJRA, S. **Comunidades Virtuais: um fenômeno social autopoiético na sociedade do conhecimento**. Dissertação (Mestrado). PUC/SP - Brasil, 2002. Disponível em: <www.tajratecnologias.com.br/artigo4.htm>. Acesso em: 19 ago. 2011.

TeHospedo. **Hospedagem JSP com Tomcat**. Disponível em: <http://tehospedo.com.br/tomcat>. Acesso em: 30 nov. 2010.

TEIXEIRA, F. A. G.; Duque, C. G. **A Recuperação da Informação e a colaboração de usuários na Web – Novas oportunidades para a Comunicação**. In: II Congreso Internacional Comunicación 3.0. Salamanca. Espanha. 2010. Disponível

em: <http://campus.usal.es/~comunicacion3punto0/comunicaciones/047.pdf>. Acesso em: 17 jun. 2011.

Testar.me. **Introdução ao Teste de Desempenho e software**. 2010. Disponível em: <www.testar.me/pages/Introducao_ao_Teste_de_Desempenho_e_software.doc>. Acesso em: 20 set. 2011.

The Apache Software Foundation. **JMeter - User's Manual: Glossary**. 2011. Disponível em: <http://jakarta.apache.org/jmeter/usermanual/glossary.html>. Acesso em: 18 set. 2011.

_____. **Apache Tomcat**. 2011. Disponível em: <http://tomcat.apache.org>. Acesso em: 01 set. 2011

TOMAÉL, M. I. **Redes sociais, conhecimento e inovação localizada**. Inf. Inf., Londrina, v. 12, n. esp., 2007. Disponível em: < http://www.uel.br/revistas/uel/index.php/informacao/article/download/1782/1519>. Acesso em: 21 mar. 2011.

VELOSO, S. Código fonte original do analisador em português. Disponível em: < http://www.devmedia.com.br/JavaMagazine/downloads/download_ed/edi_49/jm-lucene.rar>. Acesso em: 26 set. 2011.

_____. Conhecendo o Apache Lucene. Java Magazine 49. DevMedia. 2008. Disponível em: <http://www.devmedia.com.br/articles/viewcomp.asp?comp=8308>. Acesso em: 27 set. 2011.

XAVIER, C. O.; CARVALHO L. C. **Desenvolvimento de Aplicações Sociais a partir de APIs em Redes Sociais Online**. Relatório Técnico. 2011. Disponível em: < http://www.inf.ufg.br/this2/uploads/files/1/RT-INF_001-11.pdf>. Acesso em: 23 mai. 2011.