UNIVERSIDADE DE SÃO PAULO

Escola de Artes, Ciências e Humanidades

Marco Altran Ruiz

Interface para dispositivo móvel para operações OLAP em dados governamentais

São Paulo

Janeiro de 2013

Universidade de São Paulo

Escola de Artes, Ciências e Humanidades

Marco Altran Ruiz

Interface para dispositivo móvel para operações OLAP em dados governamentais

Monografia apresentada à Escola de Artes, Ciências e Humanidades, da Universidade de São Paulo, como parte dos requisitos exigidos na disciplina ACH 2018 – Projeto Supervisionado ou de Graduação II, para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Luciano Antonio Digiampietri

Modalidade:
TCC Longo (1 ano) – individual

São Paulo

Janeiro de 2013

Dedicatória

Dedico este trabalho ao meu orientador, Prof. Dr. Luciano Antonio Digiampietri, por ter sido sempre presente e ativo em me ajudar a concluir este trabalho, além de ser um exemplo de ótimo professor.

Glossário

dp: *density-ndependent pixels* – um *dp* corresponde a um pixel em 160 *dpi*.

dpi: *dots per inch* – equivalente ao pontos por polegada.

engine: motor utilizado pelo SGBD.

GB: GigaBytes – unidade de medida de armazenamento computacional.

Índices full-text: Índice utilizado em tipos de dados com tamanhos variáveis, como por exemplo, do tipo texto.

OLAP: *On-line Analytical Processing* – capacidade para manipular e analisar um grande volume de dados sob múltiplas perspectivas.

Open Data: ou dados abertos, conceito de que o dado pode ser livremente utilizado por qualquer pessoa, sem restrições.

Relative Layout: classe *Android* que padroniza parâmetros do *layout*, utilizando sempre as posições relativas para determinar o aspecto visual.

SGBD: Sistema de Gerenciamento de Banco de Dados – conjunto de programas de computador responsáveis pelo gerenciamento de uma base de dados.

SICs: Serviços de Informação ao Cidadão – Sistema governamental que permite que qualquer pessoa, física ou jurídica, encaminhe pedidos de acesso a informação para órgão e entidades do Poder Executivo Federal.

SQL: *Structured Query Language* – linguagem de pesquisa declarative padrão para banco de dados relacional.

Web Service: solução utilizada na integração de sistemas e na comunicação entre diferentes aplicações.

WYSIWYG: *What You See Is What You Get* – capacidade de um programa de computador de permitir que um documento, enquanto manipulado na tela, tenha a mesma aparência de sua utilização, usualmente sendo considerada final a forma impressa.

Resumo

Este projeto visa a desenvolver um aplicativo móvel que manipule dados governamentais, transformando dados puros em representações gráficas. As informações manipuladas foram obtidas através das despesas governamentais, que estão sendo liberados em formatos abertos desde que a Lei do Acesso à Informação foi decretada. O aplicativo será integrado via *Web Service* e um Sistema de Gerenciamento de Banco de Dados (SGBD), que foram desenvolvidos paralelamente. Com este sistema, foi possível efetuar operações *On-line Analytical Processing* (*OLAP*) a partir de dados governamentais e promover diversas visualizações, que são geradas conforme o usuário necessitar.

Palavras chaves: Lei de Acesso à Informação, *OLAP*, Sistemas integrados, Dados abertos, Dados governamentais.

Abstract

This project was to develop a mobile application that could manipulate government data, transforming raw data into graphical representations. Those manipulate data was obtained using government expenses, that is being released in open data format since the law “Lei do Acesso à Informação” went to action. This application was integrated within a Web Service and a Data Base Management System that was developed at the same time. With this system, it was possible to make On-line Analytical Processing (OLAP) operation with those informations and allow several types of visualization, which was generated at the user’s request.

Keywords: Lei de Acesso à Informação, OLAP, Integrated Systems, Open data, Government Data.

Sumário

1 Introdução

Com a lei nº 12.527, de 18 de novembro de 2011, foi regulamentado o acesso à informação pública no Brasil, que obriga órgãos públicos a prestarem informações sobre suas atividades a qualquer cidadão interessado. Após essa data, foram criados diversos Serviços de Informação ao Cidadão (SICs) para atender e disponibilizar informações públicas à população. Serviços como Portal da Transparência, criado pelo governo federal, e Portal do Cidadão, criado pelo governo estadual de São Paulo, disponibilizam informações em formatos abertos (*Open Data*) que podem ser manipuladas e categorizadas sem intervenção humana. Ou seja, a partir de um arquivo em formato aberto, que contém informações públicas, é possível criar gráficos e gerar associações usando somente processamento digital.

A partir deste panorama, surgiu a ideia do desenvolvimento de um aplicativo para dispositivos móveis, especificamente os que possuem sistema operacional *Android*[1], que possa processar informações públicas, especificamente gastos públicos, disponibilizando-as ao usuário de forma simplificada e relevante, a fim de facilitar a fiscalização dos órgãos públicos pelos cidadãos. O aplicativo funciona como uma ferramenta para quem procura informações, desde o valor agregado do gasto realizado nos munícipios de São Paulo, até análises do valor orçamentado e o valor gasto por cada departamento, ordenados do maior para o menor. Dessa forma, o usuário pode entender com mais facilidade quanto do dinheiro público está sendo gasto e onde.

O cenário atual não é o ideal, pois ainda há falta de informação e, mesmo quando a informação está disponível, há falta de confiabilidade e padronização. Para o bom uso de dados governamentais, essas informações precisam ser disponibilizadas em formato padronizado, aberto e acessível (GERMANO, E. C; TAKAOKA, H., 2012, p. 1). Apesar destas limitações, já existe uma gama bastante significativa de informações disponíveis, porém, desde a criação dos SICs, essa nova fonte de dados está sendo subutilizada, já que a disponibilização dessas informações não é de fácil leitura. A partir deste projeto, o cidadão

[1] http://www.android.com/

terá a opção de acessar algumas dessas informações via gráficos e análises, com filtros por departamentos, regiões, data, etc.

2 Objetivos

2.1 Objetivo Geral

O objetivo geral deste trabalho foi desenvolver uma ferramenta de visualização de dados governamentais, de fácil utilização e que possibilite a qualquer usuário que não tenha conhecimento técnico fiscalizar os órgãos públicos. O foco deste trabalho foi de manipular informações obtidas de despesas governamentais, que foram disponibilizadas em formato aberto através dos SICs. O objetivo foi oferecer uma aplicação que realize operações *OLAP*, onde o usuário tem um controle maior sobre os dados, tendo o poder de agregar e requisitar informações conforme ele precise. O escopo deste projeto é somente criar uma ferramenta de apoio à decisão, sem criar interpretação ou conclusão a respeito dos dados. Porém, para auxiliar na tomada de decisão serão desenvolvidos mecanismos de consolidação dos dados e identificação de valores atípicos (*outliers*).

2.2 Objetivos Específicos

Os objetivos específicos deste trabalho foram:

- Criar uma ferramenta *WYSIWYG* (*What You See Is What You Get*);
- Oferecer filtros de seleção que sejam relevantes ao usuário, e em seguida fornecer os resultados por meio de gráficos e estatísticas de gastos;
- Criar um sistema integrado que seja escalonável para utilização de outras fontes de dados;
- Fornecer uma aplicação que consiga em tempo aceitável disponibilizar os resultados da pesquisa, oferecendo uma boa experiência ao usuário;
- Identificar *outliers*.

3 Trabalhos Relacionados

Em (BICO et al, 2012), é discutido a diminuta legibilidade dos dados da Câmara Municipal de São Paulo, o que dificulta a análise direta da informação nele contida. Foi criado um modelo de dados multidimensional e hierárquico Snow Flake Schema, utilizando dados referentes à Dotação Orçamentária, que foram disponibilizadas em formato XML. A partir desse modelo, foi possível gerar gráficos e relatórios com base nos filtros escolhidos, a fim de analisar a informação com maior facilidade. O artigo conclui que a forma como os dados são disponibilizados é um grande problema, inclusive que o formato XML não é a melhor opção para se manipular dados. Esse estudo foi primordial para que pudéssemos escolher nossa fonte governamental.

4 Metodologia

A metodologia deste trabalho consistiu, primeiramente, na revisão bibliográfica descrita na seção 3 a fim de obter trabalhos com propostas semelhantes, e conhecer as abordagens utilizadas. Após esta etapa, foi realizado o levantamento de dados de fontes governamentais, que nos ajudou a decidir a base de dados mais apta para os objetivos deste projeto.

4.1 Levantamento de Dados de Fontes Governamentais

Um dos principais desafios foi encontrar a fonte governamental para obtenção de dados de despesas referentes aos estados e municípios do Brasil. Cada esfera estadual é responsável por criar seu próprio portal, que irá fornecer informações a seus cidadãos. Os portais não são padronizados e, aparentemente, estão em estágios iniciais de desenvolvimento. Para diminuir a complexidade deste trabalho, restringimos o escopo para utilização de dados obtidos

somente a partir do Portal do Cidadão do Estado De São Paulo[2]. Através dele foi possível descarregar todos os arquivos referentes às despesas dos municípios de São Paulo utilizando um *script shell*[3] no dia 25 de maio de 2013. Do estudo exploratório realizado, este portal foi o mais bem cotado nos quesitos dos dados serem: padronizados, confiáveis e quantitativos.

Em outros portais, como no Portal da Transparência do Governo Federal, é possível obter informações detalhadas referentes às despesas do governo federal, mas não é possível copiar essa informação em formato aberto[4]. O Tribunal de Contas do Estado do Rio Grande do Sul disponibilizou, com eficiência, os balancetes de despesas de seus órgãos municipais[5]. Porém entre os portais, nota-se uma falta de padronização dos arquivos disponibilizados, acarretando na tarefa de mapeamento das colunas (referentes aos dados) para que seja possível a utilização de múltiplas fontes de dados. Por essa razão, excluímos momentaneamente esse portal como uma fonte de dados para este projeto.

Outros estados também possuem seus programas de transparência, mas como as informações não estão centralizadas torna-se uma difícil tarefa reunir informações de todos os estados e municípios. O Portal Brasileiro de Dados Abertos[6] foi feito pelo governo no intuito de centralizar todas as informações governamentais em formato aberto. Porém, diversas informações são encontradas somente nos portais estaduais.

4.2 Problemas dos Dados Utilizados

No caso dos arquivos adquiridos no Portal do Cidadão do Estado de São Paulo, os campos são separados por ponto e vírgula, quando deveriam ser separados somente por uma vírgula,

[2] http://www.portaldocidadao.tce.sp.gov.br

[3] Autoria de Andre Kawamura Oliveira. Disponível em https://www.dropbox.com/s/pkjwqjcd1o0k4ud/pega_todas_despesas.sh

[4] http://www.portaltransparencia.gov.br/despesasdiarias

[5] http://dados.tce.rs.gov.br/dados_siapc.html#empenhos

[6] http://dados.gov.br

de acordo com os padrões *Comma-Separated Values* (*CSV*) descrito no informe RFC 4180[7]. Outro problema foi encontrar inconsistências nos dados, como: campos com valores negativos referentes ao custo da despesa, e campos nulos em atributos onde o valor deveria obrigatoriamente existir. No entanto, não tivemos que criar um modelo de dados, como feito em (BICO et al, 2012), pois a fonte governamental escolhida já estava modelada. Só foi necessário algumas alterações na modelagem para otimizar o desempenho do banco de dados, para que pudéssemos efetuar consultas em tempo hábil. Esse processo é descrito na Subseção 4.7. Outros problemas relacionados à inconsistência dos dados são descritos na Subseção 5.1.

4.3 Tecnologias Utilizadas

Este projeto necessitou da integração de diversos sistemas para seu funcionamento. Ele segue o modelo cliente-servidor, onde o cliente utiliza o sistema operacional *Android*, e o servidor utiliza o *Apache Tomcat* 7.0[8]. A plataforma de desenvolvimento utilizada foi o Eclipse[9], utilizando a linguagem de programação *Java*[10]. Para o cliente, foi utilizado o plug-in *Android Development Tools* (*ADT*)[11] em conjunto com o *Android SDK*[12]. Também foram utilizados as bibliotecas *Gson*[13], para comunicação cliente-servidor, *Java Servlets*[14], para o recebimento de requisições vindas do cliente, e *GraphView*[15] para visualização de gráficos

[7] http://tools.ietf.org/html/rfc4180

[8] http://tomcat.apache.org/tomcat-7.0-doc/index.html

[9] http://www.eclipse.org/

[10] http://www.java.com/pt_BR/

[11] http://developer.android.com/tools/sdk/eclipse-adt.html

[12] https://developer.android.com/sdk/index.html

[13] https://code.google.com/p/google-gson/

[14] http://docs.oracle.com/javaee/6/api/javax/servlet/package-summary.html

[15] http://www.jjoe64.com/p/graphview-library.html

dentro do aplicativo móvel. O SGBD escolhido foi o *MySQL*[16], e a transformação dos dados obtidos na Subseção 4.1 foi auxiliada pela ferramenta *Pentaho Spoon*[17].

4.4 Levantamento de Requisitos de Software

4.4.1 Requisitos Funcionais

Requisitos referentes à aplicação para dispositivos móveis (cliente):

- R1 - A aplicação deve funcionar para as versões *Android* 2.3 ou superior.
- R2 - A aplicação deverá conectar-se ao *Web Service* via *JSON*[18].
- R3 - A aplicação deve permitir o usuário comentar a respeito de um gráfico. O usuário deve ser capaz de visualizar comentários de outros usuários, mas somente irá visualizar aqueles que comentaram e utilizaram os mesmos filtros para geração do gráfico.
- R4 - A aplicação deve apresentar gráficos de acordo com os filtros selecionados pelo usuário. O gráfico será obtido após a seleção de filtros feita pelo usuário e após o resultado da requisição obtida do *Web Service*.
    - R5 - A aplicação deve permitir que o usuário visualize a mesma informação (ou seja, a mesma seleção de filtros) referente às despesas e orçamentos, de modo que seja possível alterna-los com um clique.
    - R6 - A aplicação deve fornecer uma comparação entre os dados referentes às despesas e orçamentos.

[16] http://www.mysql.com

[17] http://community.pentaho.com/

[18] http://www.json.org/

- R7 - Para cada gráfico gerado e apresentado para o usuário, a aplicação deverá gerar legendas que definem o que está sendo visualizado.

- R8 - A aplicação deve capacitar o usuário de compartilhar o gráfico que for apresentado a ele. Haverá um botão presente sempre que a aplicação apresentar um gráfico. Ao apertar esse botão, o usuário terá a opção de compartilhar a informação através de qualquer aplicativo de comunicação instalado no dispositivo do usuário (como por exemplo, redes sociais ou e-mail). Dentro do aplicativo escolhido, serão apresentadas as seguintes informações: *URL* referente ao gráfico, descrição dos filtros selecionados e *URL* para *download* do aplicativo. O usuário poderá revisar e alterar essas informações antes de publicar ou enviar para um amigo.

- R9 - Haverá uma área dentro da aplicação onde usuários poderão publicar análises feitas a partir dos gráficos gerados. O administrador deve autorizar a análise para que ela possa ser publicada.

Requisitos referentes à aplicação para o servidor:

- R10 - O servidor deverá receber e processar informações do cliente através de *Java Servlets*, que estará integrado com o SGBD.

- R11 - Ao receber uma requisição do cliente para geração de um gráfico, o servidor deve consultar a *API* disponibilizada pelo governo, se houver. Se a *API* não estiver disponível ou falhar, o *Servlet* deve consultar o SGBD.

- R12 - O servidor deve processar e salvar os comentários no SGBD. Junto com o comentário, serão gravados os filtros utilizados, para especificar quando devem ser visualizados.

- R13 - Ao receber uma requisição do cliente para geração de um gráfico, o servidor deve retornar a *URL* referente ao gráfico, e um arranjo com os comentários publicados com o mesmo conjunto de filtros.

- R14 - Se o servidor não conseguir obter informações suficientes para obtenção de um gráfico, deverá retornar ao cliente uma mensagem descrevendo o erro gerado.

O servidor também deve gerar e salvar um arquivo log contendo as seguintes informações:

- Data do erro;
- Descrição do erro;
- Filtros utilizados que geraram o erro.

4.4.2 Requisitos Não-Funcionais (RNFs)

- Requisitos de confiabilidade:
  - RNF1 - Os dados originais que serão usados como fontes, devem ser extraídos de alguns dos e-SICs disponíveis para consulta de dados governamentais.
  - RNF2 - Os dados obtidos pelos órgãos públicos só devem ser modificados caso apresentem erros de semântica.
- Requisitos de interface:
  - RNF3 - A interface deve ser intuitiva e amigável.
  - RNF4 - A interface deve possuir um bom número de elementos de interação.

4.5 Critérios Ergonômicos

De acordo com (SCAPIN; BASTIEN, 1993, p. 1), o design do critério ergonômico procura "desenvolver métodos e ferramentas que irão incorporar fatores humanos no processo de design e avaliar as interfaces humano-computador" (tradução nossa). Os seguintes critérios ergonômicos foram usados no desenvolvimento deste projeto:

- Condução
  - Convite/Presteza:

        - Todos os campos conterão rótulos, informando o usuário da sua funcionalidade;
        - Os gráficos conterão legendas nos eixos.
    - Agrupamento e distinção entre itens:
        - Os filtros de pesquisa para geração do gráfico serão divididos em grupos, e para cada filtro os valores contidos serão organizados por ordem alfabética. Um critério lógico será definido para a sequência dos filtros. Filtros mais específicos estarão dentro do grupo "Mais opções" e só irão ser visualizados caso o usuário clique em "Mostrar mais opções".
    - *Feedback* imediato:
        - Toda requisição que exija conexão com a internet, deverá haver um componente gráfico que irá informar visualmente que a requisição está sendo processada;
        - Para requisições que exigem alto processamento, o usuário será informado percentualmente, quanto que uma requisição já foi processada.
- Controle explícito:
    - Controle do usuário:
        - O usuário poderá a qualquer momento interromper e reiniciar uma requisição demorada;
        - O usuário poderá recuar e avançar nas telas que já foram visitadas.
- Gestão de erros:
    - Qualidade das mensagens de erro:

- Quando for feita uma requisição e essa falhar, a aplicação móvel deve informar o erro compreensível a um usuário que não tem conhecimento dos termos técnicos em informática.

4.6 Padrões de Projeto (*Design Patterns*)

A utilização do *Android* como sistema operacional possibilita a este projeto alcançar usuários de diversos dispositivos móveis. Uma das principais diferenças entre os dispositivos está no tamanho e resolução da tela. Para criar interfaces gráficas agradáveis tanto para usuários de um pequeno *smartphone* como usuários de um *tablet*, a empresa Google desenvolveu uma vasta documentação a respeito de design em múltiplos dispositivos[19].

A aplicação irá usar o *Relative Layout*[20] como layout principal. Ele organiza os objetos visuais de acordo com suas posições relativas. O *Relative Layou*t permite, por exemplo, colocar um objeto A no centro da tela, e o objeto B logo abaixo de A. Dessa maneira, não importa se a resolução do dispositivo é baixa ou alta, os objetos estarão posicionados da mesma maneira (em suas devidas posições relativas).

É muito comum definir o tamanho de um objeto por quantidade de *pixels*. Se a interface gráfica fosse usada dessa maneira e criássemos um *layout* satisfatório para um dispositivo de alta resolução, o mesmo layout não iria ser visualizado por inteiro em um dispositivo de baixa resolução. Por essa razão, foi utilizada a unidade *density-independent pixels* (*dp*) para definir o tamanho de um objeto. Uma unidade *dp* equivale ao valor físico de um pixel em um dispositivo com densidade de 160 *dpi* (*dots per inch*, ou pontos por polegada)[21]. Dessa maneira, é possível desenhar objetos que são escalonáveis para a maioria dos dispositivos.

As figuras de 1 a 5 ilustram a interface gráfica que é fornecida ao usuário.

---

[19] http://developer.android.com/guide/practices/screens_support.html

[20] http://developer.android.com/guide/topics/ui/layout/relative.html

[21] http://developer.android.com/training/multiscreen/screendensities.html

Figura 1: Seleção de filtros.

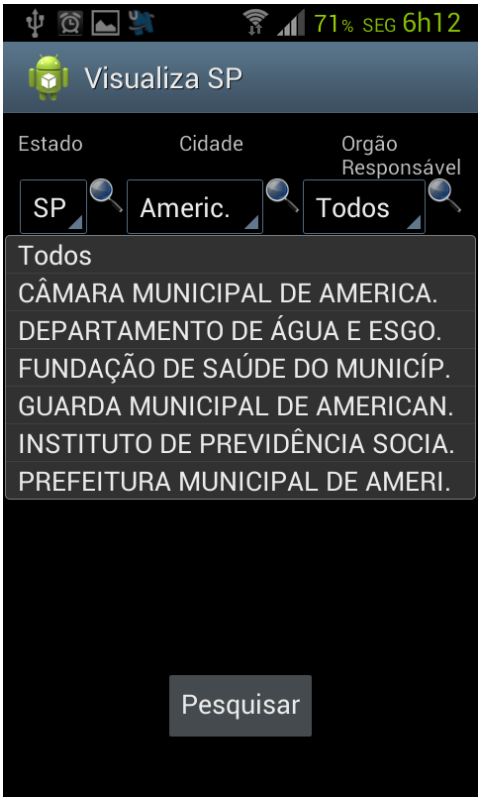

Figura 2: Seleção de filtros.

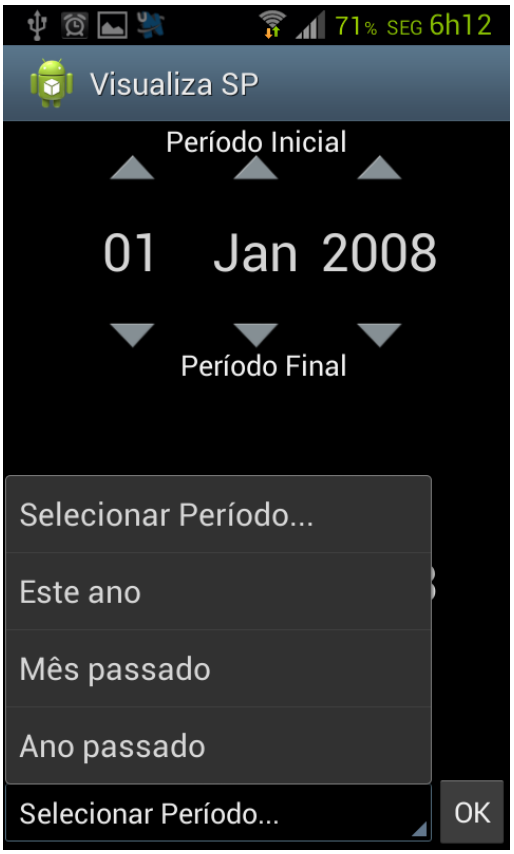

Figura 3: Seleção do período.

4.7 Otimização do Banco de Dados

Pelo grande volume de dados obtidos através da fonte governamental utilizada, o banco de dados criado ficou com um grande volume de dados. O tamanho total dos arquivos descarregados (descritos na Subseção 4.1) foi de 3,60 Gigabytes (GB), 54,60 GB quando descompactados[22]. Para que as consultas fossem feitas em tempo aceitável para o usuário, houve uma grande necessidade de otimizar o banco. Alguns dos maiores desafios enfrentados por este TCC estão relacionados à otimização do banco de dados, por isso as atividades realizadas para esta seção serão descritas detalhadamente.

Algumas medidas foram utilizadas a fim de reduzir o tempo das consultas. Como o SGBD utilizado foi o *MySQL* utilizando o motor de execução (*engine*) *InnoDB*, procuramos entender suas especificidades, para que pudéssemos aplicar as técnicas mais eficientes para o nosso objetivo.

Foi criada uma tabela "DESPESAS" utilizando a ferramenta *Pentaho Spoon*, onde toda a informação relacionada às despesas foi mantida nessa tabela. Para essa tabela foram criados:

1. O índice "FTS_DOC_ID" como chave primária, utilizando as propriedades "BIGINT", "UNSIGNED", "NOT NULL" e "AUTO_INCREMENT". Essas propriedades foram utilizadas pelo fato da chave "FTS_DOC_ID" ser um nome reservado do *MySQL*, que o utiliza internamente para lidar com índices *full-text*[23], utilizados nos itens 2.c;

2. Índices secundários para os atributos que estavam sendo mencionadas nas consultas SQL (*Structured Query Language*) com as cláusulas "WHERE", "GROUP BY" ou "ORDER BY". Para cada atributo enquadrado nessa situação, adotou-se a seguinte estratégia:

[22] Os arquivos são disponibilizados em formato ZIP porque o conteúdo dos documentos CSV contém uma alta taxa de compressão, resultando em um arquivo de tamanho final bem reduzido do original. Justamente por causa disso, é discutido na Subseção **Erro! Fonte de referência não encontrada.** a motivação de comprimir o banco de dados como mais um método para otimização.

[23] https://blogs.oracle.com/mysqlinnodb/entry/innodb_full_text_search_performance

a. O atributo que continha campos com valores numéricos ou do tipo “Date”, e que utiliza os operadores de comparação =, >, >=, <, <=, ou “BETWEEN” utilizou um índice do tipo árvore B;

b. O atributo que necessitava somente do operador = utilizou um índice do tipo *hash* (que só permite o uso deste operador) e que é mais rápido se comparado ao índice do tipo árvore B[24]. Como esse índice não otimiza a cláusula “ORDER_BY”, caso o atributo faça uso de tal cláusula, o índice escolhido foi do tipo árvore B[25];

c. Os atributos do tipo “CHAR”, “VARCHAR”, ou “TEXT” utilizaram um índice do tipo *full-text*. Como esse tipo de índice é o mais lento dentre os três, ele só é utilizado caso não haja outro índice alternativo com outra tipagem.

O índice *full-text* é uma das especificidades do *MySQL*. Nem todos os SGBDs permitem indexar atributos do tipo “CHAR”, “VARCHAR”, ou “TEXT” como o *full-text* faz. Embora ele seja o mais lento dentre os três, nossos resultados (contidos na Subseção 5.3) mostram que ele ainda é bem mais eficaz do que um atributo que não utiliza índice. Utilizamos a sintaxe “MATCH() AGAINST()”[26] para utilizar um índice *full-text*, que aceita os modificadores “IN BOOLEAN MODE” e “IN NATURAL LANGUAGE MODE”. Escolhemos o primeiro modificador, por ele ter executado as nossas consultas em menor tempo, e pelo segundo modificador ignorar as palavras mais frequentes o que, consequentemente, trouxe resultados indesejados (trazendo resultados além dos requisitados).

Os arquivos fonte já continham diversos atributos que funcionavam como índices para outros atributos, esses geralmente do tipo texto. Por exemplo, o atributo “ds_programa” contém o nome do programa vinculado a uma despesa, e o atributo “cd_programa” contém um valor inteiro que corresponde a um índice do nome do programa. Dessa forma, é muito mais rápido: (1) procurarmos o código a que pertence o nome do programa desejado, e

[24] http://dev.mysql.com/doc/refman/5.5/en/index-btree-hash.html

[25] http://dev.mysql.com/doc/refman/5.5/en/index-btree-hash.html

[26] http://dev.mysql.com/doc/refman/5.0/en/fulltext-search.html

fazermos a consulta utilizando o atributo “cd_programa” (que usa um índice *hash*, o mais rápido); do que (2) não procurarmos o código pertencente e realizarmos nossa consulta utilizando o atributo “ds_programa” (que usa um índice full-text, o mais lento). Porém, notamos uma grande existência de um atributo equivalente ao “cd_programa” para o atributo “ds_municipio”, que contém o nome da cidade e era utilizado na maioria de nossas consultas. Para isso criamos o atributo “cd_municipio”. Primeiramente utilizou-se o método “ALTER TABLE” para adicionar um novo campo a uma tabela existente. Pelo banco ser bastante grande, essa tarefa era extremamente custosa e impossível de ser concluída: ou por demorar tempo demasiadamente grande (na ordem de dias ou mais), ou por barrar em questões físicas, como falta de memória *RAM* ou espaço em disco. Na segunda tentativa, recriamos o banco utilizando os atributos originais, adicionando a chave primária (descrita no item 1 desta Subseção) e o novo atributo “cd_municipio”. Feito isso, realizamos as seguintes etapas:

3. Com o auxílio da ferramenta *Pentaho Spoon*, fizemos a transformação dos dados *CSV* para a tabela[27] recém-criada. O atributo “cd_municipio” manteve-se nulo, e consequentemente, não foi populado nessa etapa. O processo durou cerca de 9 horas[28];

4. Após a conclusão da etapa 3, foi criada uma nova tabela chamado “lista_cidades” que contém, em cada registro, o nome de uma cidade e seu id, incluindo todas as cidades disponíveis no banco. Esse processo durou cerca de 18 minutos. Esse tipo de requisição era realizada sempre que o usuário deseja selecionar seus filtros. Se essa etapa não fosse realizada, seria inviável para um usuário ter de esperar 18 minutos toda vez que ele necessitasse requisitar uma informação;

```
drop table if exists lista_cidades;

create table IF NOT EXISTS lista_cidades(ds_municipio TEXT, id SMALLINT UNSIGNED
NOT NULL AUTO_INCREMENT PRIMARY KEY );

truncate table lista_cidades;
```

[27] Código *SQL* usado para criar a tabela mencionada: https://www.dropbox.com/s/v7ocye2x90gy0ma/create_table_despesas.sql

[28] Essa etapa e as demais utilizaram um PC com a seguinte configuração: AMD Phenom II X4, 8GB RAM, Windows 7 64bit, SSD 128GB.

```
insert into lista_cidades(ds_municipio)

select distinct ds_municipio from despesas;
```

5. Em seguida, foi realizado o mapeamento do novo atributo “cd_municipio” onde para cada registro, era inserido no “cd_municipio” o id de “ds_municipio” localizado na tabela “lista_cidades”. O processo durou cerca de 4 dias para ser concluido;

```
update despesas,lista_cidades

set despesas.cd_municipio = lista_cidades.id

where despesas.ds_municipio = lista_cidades.ds_municipio;
```

6. Por último, foram criados todos os índices secundários necessários[29], de acordo com o item 2 desta Subseção.

Além das medidas mencionadas, foram feitos ajustes de configuração no banco (*database tuning*) para otimizar os recursos físicos disponíveis. As principais modificações que ocasionaram efeitos positivos na duração do processamento das consultas foram: (1) aumento do valor da variável de configuração “innodb_buffer_pool_size”, que declara quanto de memória será usado como *buffer* para manter um *cache* dos dados e índices, e (2) utilização de um *cache* que guarda os resultados das consultas (*queries*) realizadas. Pode-se ver na Subseção 4.8 que o resultado das consultas demanda muito processamento, mas o tamanho da resposta é relativamente pequeno, pela consulta retornar valores agregados. Guardar os resultados em um *cache* é muito vantajoso, pois nessa situação um resultado irá ocupar pouco espaço em memória e irá evitar que o mesmo processamento seja feito várias vezes.

4.8 Outras Otimizações

Além das otimizações realizadas no banco de dados, ainda há uma camada *Java* no servidor para efetuar o processamento restante necessário. Ao longo do desenvolvimento

[29] Código *SQL*: https://www.dropbox.com/s/zamwd94ms7jtbzw/create_index.sql

diversas adaptações foram realizadas para se encontrar um balanço que distribuía a carga entre a camada *Java* e a camada do SGBD, a fim de obter os menores tempos de processamento.

Após o usuário selecionar os filtros para sua pesquisa, uma série de processamentos devem ser realizados para devolver como resultado: (1) um gráfico contendo o valor gasto distribuído no tempo e (2) os 10 programas, órgãos, responsáveis, etc., que tiveram os maiores gastos. Na figura 4 e 5, é possível ver um exemplo de um resultado. Utilizamos duas abordagens para alcançar esse resultado, sendo o último o escolhido por ser mais eficiente. Utilizando um exemplo de que o usuário deseja saber informações referentes às despesas de Osasco no ano de 2012, a primeira abordagem utilizou as seguintes etapas:

1. É realizada a seguinte consulta *SQL*:

2. É criada uma estrutura de dados de tamanho igual ao total de dias contidos no período definido no filtro (no exemplo, o tamanho total é de 365). Cada campo contém a somatória dos valores gastos a data que está atrelada àquele índice. Com essa estrutura, já é possível plotar o gráfico, onde cada campo equivale ao total gasto em uma determinada data.

3. Para cada categoria (programa, órgão, etc.) é criada uma estrutura dinâmica. Para cada registro, é somado o seu valor da despesa em cada categoria pertencente aquele registro. Depois essa estrutura é ordenada do maior para o menor utilizando o algoritmo *Quicksort*, a fim de obter os 10 maiores gastos de cada categoria.

Figura 4: Exemplo de resultado.

Figura 5: Exemplo de resultado.

Pensamos que poderíamos entregar o mesmo resultado de maneira mais eficiente, acabando por adotar uma segunda abordagem, que realiza as principais operações utilizando somente a camada do SGBD:

1. É realizada a seguinte consulta *SQL* para a construção do gráfico:

   Select dt_emissao_despesa, sum(VL_DESPESA) AS VL_DESPESA From despesas

   where cd_municipio=(Select id from lista_cidades where lista_cidades.ds_municipio='Osasco' LIMIT 0,1)

   AND ano_exercicio=2012 GROUP BY dt_emissao_despesa ;

2. É criada uma tabela temporária com as informações necessárias para obter os 10 maiores gastos de cada categoria:

   CREATE TEMPORARY TABLE IF NOT EXISTS table2 AS (Select ds_despesa, ds_orgao, ds_fonte_recurso, ds_programa, tp_despesa, sum(VL_DESPESA) AS vl_total

   from despesas where cd_municipio=(Select id from lista_cidades where lista_cidades.ds_municipio='Osasco' LIMIT 0,1) AND ano_exercicio=2012

   group by ds_despesa, ds_orgao, ds_fonte_recurso, cd_programa, tp_despesa);

5 Resultados

5.1 Inconsistência dos Dados

Pela fonte de dados utilizada se tratar de despesas, e por haver uma fonte de dados específica para receitas, acreditamos que a maioria dos valores de despesa declarados como negativo, foram declarados de forma inconsistente. Dos 123,31 milhões de declarações de despesas contidas no nosso banco de dados, 2,96 milhões contêm valores negativos, o que corresponde a 2,4% das despesas totais. Isso significa que os valores apresentados neste projeto podem ter deixados de contabilizar um total de até R$38,95 bilhões, somente por conta da inconsistência das despesas negativas.

Observou-se também a inconsistência dos índices presentes na fonte de dados. Há diversas colunas que são derivadas de outras colunas. Por exemplo, a coluna “ano_exercicio” e “mes_referencia” são derivadas da coluna “dt_emissao_despesa”, que contém a data completa da despesa. Em uma despesa, se o campo contido na coluna “dt_emissao_despesa”, por exemplo, for “2011-03-29 00:00:00”, e o campo contido na coluna “ano_exercicio” for diferente de 2011, há uma inconsistência na informação apresentada. Dos 123,31 milhões registros de despesas, 3,73 milhões possuem divergências entre a data contida na coluna “dt_emissao_despesa” e a coluna “ano_exercicio” ou “mes_referencia”. Isso infere que 3,02% dos índices relacionados à data são inconsistentes. Outras colunas funcionam como índices, como por exemplo, “cd_programa” é um índice de “ds_programa”. Em algumas consultas, foi observadas divergências nos resultados dados pela coluna “cd_programa” e “ds_programa”.

5.2 Outliers

Um dos objetivos deste projeto era a descoberta de *outliers*, resultados que fogem dos padrões de outros resultados dentro da mesma categoria. Pelos dados serem visualizados a partir de um gráfico, é fácil identificar visualmente alguns resultados discrepantes.

Em um dos nossos testes realizados, filtrando somente para o município de Santo André, e descrição da despesa como “BOLSA DE ESTUDOS ATLETAS” notamos que o número de despesas informados juntamente com o montante total gasto do ano de 2012 é incompatível com os padrões dos outros anos, vide tabela 1.

Ano do exercício	Número de despesas informado	Montante total gasto
2008	49470	R$835,59 milhões
2009	47877	R$851,18 milhões
2010	54860	R$938,58 milhões
2011	53771	R$1,1 bilhão

2012	467	R$25,68 milhões

Tabela 1: resultado das consultas.

Um relatório[30] de número 88 TC-001401/026/11 entregue ao Tribunal de Contas do Estado de São Paulo relativo ao munícipio de Santo André e ao exercício de ano 2011, relata no item z.2 da Subseção 1.5 que houve inserção incorreta dos dados atribuídos ao credor "Bolsa de Atletas" no sistema eletrônico, sendo o montante gerado real muito menor ao informado, e que isso iria ser corrigido em 2013, desativando este credor. Isso pode explicar a discrepância dos valores observados, porém o relatório carece de detalhes para total esclarecimento deste problema específico. Não houve menção deste problema em relatórios referentes ao exercício de ano 2012.

5.3 Desempenho

Para avaliarmos a evolução do desempenho do banco de dados ao logo do processo de otimização, realizamos 8 testes. Cada teste contém um conjunto de parâmetros, que se referem a uma seleção de filtros, vide tabela 2. Para os testes de 1 a 4, avaliamos o desempenho às consultas que retornavam o conjunto de pontos, que servirão para construir o gráfico. A fim de avaliar as otimizações feitas nos índices, para cada teste medimos o desempenho de três métodos:

1. Sem utilizações de índice no atributo "ds_municipio";
2. Com a utilização de índice full-text no atributo "ds_municipio";
3. Com a utilização de índice hash no atributo "cd_municipio".

	Tipo de consulta	Município	Período	Órgão

[30] http://www2.tce.sp.gov.br/arqs_juri/pdf/260202.pdf

Teste 1	Construir gráfico	Santo André	Todos	Todos
Teste 2	Construir gráfico	Santo André	2012	Todos
Teste 3	Construir gráfico	Santo André	Todos	PREFEITURA MUNICIPAL DE SANTO ANDRÉ
Teste 4	Construir gráfico	Santo André	2012	PREFEITURA MUNICIPAL DE SANTO ANDRÉ
Teste 5	Calcular estatística	Santo André	Todos	Todos
Teste 6	Calcular estatística	Santo André	2012	Todos
Teste 7	Calcular estatística	Santo André	Todos	PREFEITURA MUNICIPAL DE SANTO ANDRÉ
Teste 8	Calcular estatística	Santo André	2012	PREFEITURA MUNICIPAL DE SANTO ANDRÉ

Tabela 2: parâmetros dos testes.

Teste 1	Duração (em segundos)	Eficiência em relação ao resultado de cima (em percentagem)
Método 1	8277,288	-
Método 2	7,503	0,09%

Método 3	1,825	24,32%

Tabela 3: Teste 1.

Teste 2	Duração (em segundos)	Eficiência em relação ao resultado de cima (em percentagem)
Método 1	139,465	-
Método 2	6,926	4,96%
Método 3	1,825	26,35%

Tabela 4: Teste 2.

Teste 3	Duração (em segundos)	Eficiência em relação ao resultado de cima (em percentagem)
Método 1	8211,783	-
Método 2	7,738	0,09%
Método 3	3,198	41,32%

Tabela 5: Teste 3.

Teste 4	Duração (em segundos)	Eficiência em relação ao resultado de cima (em percentagem)
Método 1	145,565	-

Método 2	7,364	5,06%
Método 3	2,464	33,46%

Tabela 6: Teste 4.

Para os testes de 5 a 8, avaliamos o desempenho das consultas que calculam os 10 maiores gastos, para cada categoria. A fim de avaliar as otimizações feitas na Subseção 4.8, para cada teste medimos o desempenho de três métodos:

1. Agrupamento e ordenação feita pela camada *Java*;
2. Agrupamento e ordenação feita pela camada SGBD;
3. Agrupamento e ordenação feita pela camada SGBD, com o auxilio de uma tabela temporária.

Para os testes de 5 a 8, utilizamos sempre os melhores índices possíveis.

Teste 5	Método 1	Método 2	Método 3
Duração para criar tabela temporária (em segundos)	-	-	24,257
Duração para encontrar os 10 maiores gastos agrupados por descrição da despesa (em segundos)	-	15,013	0,790
Duração para encontrar os 10 maiores gastos agrupados por orgão	-	15,436	0,791

(em segundos)			
Duração para encontrar os 10 maiores gastos agrupados por fonte do recurso (em segundos)	-	14,257	0,788
Duração para encontrar os 10 maiores gastos agrupados por programa (em segundos)	-	11,968	0,776
Duração para encontrar os 10 maiores gastos agrupados por tipo de despesa (em segundos)	-	13,747	0,766
Duração total na camada SGBD	**8,913**	**70,421**	**28,168**
Duração total na camada Java	**52,417**	**0,037**	**0,030**
Duração total	**61,330**	**70,458**	**28,198**
Eficiência em percentagem em relação ao resultado de cima	**-**	**114,88%**	**40,02%**

Tabela 7: Teste 5.

Teste 6	Método 1	Método 2	Método 3
Duração para criar tabela temporária (em segundos)	-	-	9,228
Duração para encontrar os 10 maiores gastos agrupados por descrição da despesa (em segundos)	-	4,617	0,230
Duração para encontrar os 10 maiores gastos agrupados por orgão (em segundos)	-	4,571	0,211
Duração para encontrar os 10 maiores gastos agrupados por fonte do recurso (em segundos)	-	4,486	0,221
Duração para encontrar os 10 maiores gastos agrupados por programa (em	-	3,987	0,222

segundos)			
Duração para encontrar os 10 maiores gastos agrupados por tipo de despesa (em segundos)	-	4,412	0,214
Duração total na camada SGBD	**5,375**	**22,073**	**10,326**
Duração total na camada Java	**16,065**	**0,033**	**0,032**
Duração total	**21,44**	**22,103**	**10,358**
Eficiência em percentagem em relação ao resultado de cima	**-**	**103,09%**	**46,86%**

Tabela 8: Teste 6.

Teste 7	Método 1	Método 2	Método 3
Duração para criar tabela temporária (em segundos)	-	-	17,283
Duração para encontrar os 10 maiores gastos agrupados por descrição da despesa	-	11,880	0,550

(em segundos)			
Duração para encontrar os 10 maiores gastos agrupados por orgão (em segundos)	-	-	-
Duração para encontrar os 10 maiores gastos agrupados por fonte do recurso (em segundos)	-	11,096	0,585
Duração para encontrar os 10 maiores gastos agrupados por programa (em segundos)	-	9,447	0,563
Duração para encontrar os 10 maiores gastos agrupados por tipo de despesa (em segundos)	-	11,096	0,534
Duração total na camada SGBD	**7,756**	**43,519**	**19,515**
Duração total na camada Java	**18,677**	**0,034**	**0,033**

Duração total	**26,433**	**43,553**	**19,549**
Eficiência em percentagem em relação ao resultado de cima	**-**	**164,77%**	**44,89%**

Tabela 9: Teste 7.

Teste 8	Método 1	Método 2	Método 3
Duração para criar tabela temporária (em segundos)	-	-	8,276
Duração para encontrar os 10 maiores gastos agrupados por descrição da despesa (em segundos)	-	4,187	0,171
Duração para encontrar os 10 maiores gastos agrupados por órgão (em segundos)	-	-	-
Duração para encontrar os 10 maiores gastos agrupados por fonte do recurso (em segundos)	-	4,084	0,170

Duração para encontrar os 10 maiores gastos agrupados por programa (em segundos)	-	3,771	0,165
Duração para encontrar os 10 maiores gastos agrupados por tipo de despesa (em segundos)	-	4,080	0,160
Duração total na camada SGBD	**5,087**	**16,122**	**8,942**
Duração total na camada Java	**14,048**	**0,035**	**0,052**
Duração total	**19,135**	**16,157**	**8,994**
Eficiência em percentagem em relação ao resultado de cima	**-**	**84,44%**	**55,67%**

Tabela 10: Teste 8.

Dos testes 1 a 5, é possível notar a melhora significativa no desempenho na utilização de um dos índices citados. Nos testes 1 e 3, o método 2 necessitou somente de 0,09% do tempo utilizado pelo método 1 para efetuar o processamento. A utilização de índices tem a desvantagem de necessitar de maior espaço em disco, principalmente os índices *full-text*. Em todos os testes, o índice *hash* foi o que obteve maior desempenho.

Dos testes 6 a 8, é possível observar que atribuir uma carga maior de processamento à camada SGBD é, em média, o método mais satisfatório. O método de realizar o

processamento na camada SGBD utilizando uma tabela temporária obteve o melhor desempenho em todos os testes. A não utilização da tabela temporária desempenhou, em média, pior comparado ao método que realiza o processamento na camada *Java*. Nota-se também a importância de testar diversas formas de escrever uma consulta *SQL* que traga o mesmo resultado, por haver diferenças significativas de desempenho.

6 Discussão

Como visto na Subseção 4.1, procuramos uma fonte de dados governamental que melhor atendesse, dentre outros requisitos, a confiabilidade dos dados. Porém, como visto na Subseção 5.1, a demasiada inconsistência dos dados observados é preocupante, dificultando a utilização desses dados para qualquer fim. Para superar esta dificuldade, é necessário que os órgãos governamentais estabeleçam padrões de qualidade mais rígidos para inserção desses dados, tornando-os mais próximos aos valores reais. Outros órgãos ainda pecam em não disponibilizar suas fontes em formato CSV, e mesmo os que fazem, disponibilizam somente dados agregados. É necessário disponibilizar detalhadamente todas as despesas realizadas, tal como feito pelo Portal do Cidadão do Estado De São Paulo, para que um projeto semelhante a este seja realizado. A Lei de Acesso à Informação existe somente há alguns anos, portanto espera-se que ocorra uma evolução na qualidade dos dados disponibilizados para a população.

Além de pressionarmos os órgãos governamentais, um incentivo para surgimento e aprimoramento de projetos que trabalham com dados governamentais é essencial para que possamos entregar ferramentas úteis e benéficas à população. Órgãos governamentais podem incentivar tais projetos por meio de concursos que premiam os melhores projetos, de forma semelhante ao concurso que a prefeitura de Nova Iorque promove[31]. O código fonte deste projeto é mantido em um repositório[32], podendo ser utilizado e modificado livremente.

[31] http://nycbigapps.com

[32] https://code.google.com/p/visualizasp/

7 Conclusão

Pelo projeto ter utilizado uma alta quantidade de dados para manipulação, houve grandes desafios no quesito desempenho para alcançar os objetivos propostos. A adoção de índices, tabelas adicionais que derivam da original, novas colunas, estudos para construção eficientes de consultas *SQL*, e *database tuning* foram fundamentais para alcançar um desempenho satisfatório ao usuário. Para trabalhos futuros, o banco de dados a ser utilizado poderia utilizar a técnica de compressão disponível pelo SGBD *MySQL*. Nos estudos realizados na Subseção 4.7, houve indícios que seria benéfico utilizar uma técnica de compressão de dados, pela natureza da fonte de dados utilizada possuir um nível de compressão altamente eficiente.

A partir do trabalho realizado, foi possível encontrar evidências de irregularidades na consistência dos dados, e nas falhas de prestação de contas públicas. Caso este projeto seja apoiado por uma comunidade de programadores engajados e outros participantes, será possível oferecer um serviço de grande utilidade à população, ajudando a transparecer as contas públicas, diminuindo os níveis de desvios de verbas e aumentando a eficiência do dinheiro público gasto.

Também como trabalhos futuros, pode-se haver um foco maior na interface gráfica, dando mais flexibilidade aos desejos do usuário, já que o desempenho do sistema melhorou significativamente.

Referências Bibliográficas

BRASIL. Decreto nº 2.829, de 29 de outubro de 1998. Estabelece normas para a elaboração e a execução do plano plurianual e dos orçamentos da união, e dá outras providências. **Diário Oficial [da] República Federativa do Brasil,** Poder Executivo, Brasília, DF, 30 out. 1998, p. 9.

BRASIL. lei nº 12.527, de 18 de novembro de 2011. Regula o acesso a informações previsto no inciso XXXIII do art. 5º, no inciso II do § 3º do art. 37 e no § 2º do art. 216 da constituição federal; Altera a lei nº 8.112, de 11 de dezembro de 1990; Revoga a lei nº 11.111, de 5 de maio de 2005, e dispositivos da lei nº 8.159, de 8 de janeiro de 1991; E dá outras providências. **Diário Oficial da União**. Poder Legislativo, Brasília, DF, 18 nov. 2011. p. 1 (edição extra).

BICO et al, 2012. **Legibilidade em Dados abertos: uma Experiência com os Dados da Câmara Municipal de São Paulo**. VIII Simpósio Brasileiro de Sistemas de Informação (SBSI 2012).

GERMANO, E. C; TAKAOKA, H.. Uma Análise das Dimensões da Qualidade de Dados em Projetos de Dados Governamentais Abertos. Anais do V Congresso Consad de Gestão Pública - CONSAD, **2012**, 22 páginas.

e-SIC (Sistema Eletrônico do Serviço de Informações ao Cidadão). Disponível em < http://www.acessoainformacao.gov.br/sistema/> Acesso em 09 Mar 2013.

SCAPIN D., BASTIEN CH. **Ergonomic Criteria for the evaluation of**

human-computer interfaces. Rapport Technique INRIA N°156, 1993.