Estratégias para importação de grandes volumes de dados para um servidor PostgreSQL *

Vanessa Barbosa Rolim[1], Marilia Ribeiro da Silva[1], Vilmar Schmelzer[1]
Fernando José Braz[1], Eduardo da Silva[1]

[1]Fábrica de Software, Instituto Federal Catarinense – Câmpus Araquari

{nessabrolim, marilia.ifc, vilmarsss}@gmail.com

{fernando.braz, eduardo}@ifc-araquari.edu.br

Abstract. *Among the projects in development on Fabrica de Software environment at the Catarinense Federal Institute, one is about a traffic management support system. In this project, the import of an external, too large, and unstructured file data to an internal SQL database is required. The use of a framework Django to import this data obtained a low performance result. Then, new import strategies were desirable. Thus, this work deals with the optimization of data import strategies used to solve this problem, by presenting the proposed solutions and comparing their performance results.*

Resumo. *Dentre os projetos desenvolvidos no ambiente da Fábrica de Software do Instituto Federal Catarinense, um trata de um sistema de gestão de trânsito. No âmbito desse projeto, é necessário importar uma base de dados externa muito grande e não estruturada para uma base interna em SQL. A utilização de um framework Django para a importação dos dados resultou em baixo desempenho, iniciando a busca por novas estratégias de importação. Assim, esse trabalho trata da otimização das estratégias de importação de dados utilizadas para a resolução desse problema, apresentando as soluções propostas e comparando o desempenho dos resultados.*

1. Introdução

A Fábrica de Software, vinculada ao Núcleo de Operacionalização e Desenvolvimento de Sistemas de Informação (NODES) do Instituto Federal Catarinense, abriga, entre outros, um projeto de desenvolvimento de software em parceria com o Departamento de Trânsito de Joinville/SC (Detrans) [Mota and et al. 2014]. Um de seus objetivos trata da inconsistência e redundância de dados, visto que a base de dados utilizada atualmente pelo Detrans fica contida em uma série de arquivos de texto.

Esses arquivos se referem às características de veículos (cor, espécie e categoria), características de infração (tipo de infração, lei à qual se refere) e ao registro dos veículos em si. O conteúdo possui codificações diferentes dentro do mesmo arquivo, sendo o UTF-8 [1] a codificação predominante. Cada arquivo possui em média 50 registros, com exceção daquele que contém os dados dos veículos, doravante chamado `arquivo1`. Esse arquivo,

*Projeto de pesquisa parcialmente apoiado pelo CNPq (488004/2013-6) e do edital MEC/SETEC 94/2013

[1]Mais informações em http://www.rfc-editor.org/info/rfc3629.

no dia 20 de maio de 2015, continha cerca de 4,5 milhões de registros, ocupando 736,8 MB de espaço em disco.

Para manter a integridade e o desempenho do sistema, os dados devem ser importados para uma nova base no servidor de banco de dados. Os primeiros testes de importação realizados estimaram pelo menos uma semana para o *upload* dos dados. Esse resultado é insatisfatório para o cenário atual e projeção futura, pois é uma tarefa rotineira.

Diante disso, este artigo trata das soluções encontradas para a otimização do tempo de *upload* dos arquivos e da importação dos dados para um servidor de banco de dados. Após o término do *upload* dos registros, esse tempo passa a ser, aproximadamente, de duas horas com a otimização, que considerou fatores como a modelagem do banco de dados, a leitura de arquivos e a divisão dos dados dos arquivos por conteúdo.

Este trabalho está organizado da seguinte forma: Seção 2 descreve o cenário do projeto. Seção 3 apresenta a proposta e seus experimentos. Seção 4 apresentação dos resultados obtidos. Seção 5 conclui o artigo e apresenta perspectivas de trabalhos futuros.

2. Cenário

O `arquivo1` é um dos arquivos que contém dados sobre veículos do estado de Santa Catarina, disponibilizado pelo Detrans. Dado que o Detrans está passando por um processo de implantação de novas tecnologias, fez-se necessária a importação dos dados deste, e de outros arquivos, para um servidor de banco de dados. Esses dados serão integrados com o sistema web de Gestão de Trânsito da cidade de Joinville que está sendo desenvolvido pela Fábrica de Software.

Para a construção do projeto, são utilizados um servidor de aplicação Django 1.8, com Python 2.7, e Postgres 9.3. Para o ambiente web utiliza-se um servidor virtual sobre VMWare 5 ESXI, que roda Debian 8.0 GNU/Linux, com 4 processadores Intel Xeon de 2.13 GHz e 2 GB de memória RAM.

O `arquivo1` foi obtido através de acesso remoto utilizando o protocolo de tranferência de arquivos FTP. Além desse, outros arquivos também foram transferidos para a execução do projeto. Porém, esse artigo trata dos problemas referente ao *upload* dos dados do `arquivo1` para uma base de dados relacional.

Visto que o `arquivo1` é um arquivo de texto, a Figura 1 apresenta a sua estrutura. Pode-se dizer que sua estrutura se assemelha aos arquivos do tipo csv, porém não possui caracteres delimitadores. No `arquivo1` cada atributo de veículo possui um tamanho fixo de caracteres e, como pode ser observado na Figura 1, encontram-se armazenados de forma sequencial. Então, cada linha do arquivo deve conter o somatório de caracteres de todos os atributos, ou seja, 142 caracteres.

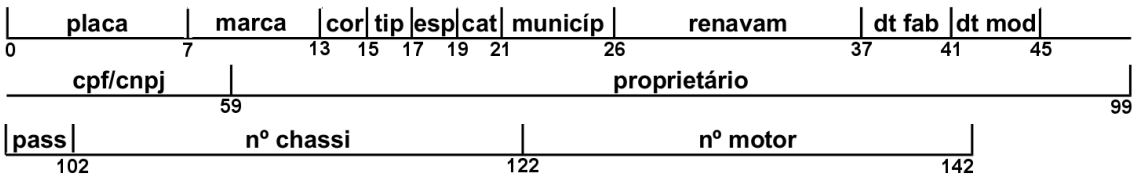

Figura 1. Formato do `arquivo1` por atributo de veículo em relação a quantidade de caracter.

Como os registros foram inseridos em um banco de dados relacional, estudou-se as características dos atributos a fim de definir um identificador único para veículo. Assim, a coluna placa não pode ser utilizada como chave primária, pois se repete indefinidas vezes no `arquivo1` que foi apontado com várias inconsistências, entre elas a codificação e a duplicidade parcial de registros. Optou-se, então, pela utilização da coluna chassi como a chave primária de veículo. As outras características tornaram-se chaves estrangeiras para o modelo relacional.

A Figura 2 ilustra a modelagem resumida do banco de dados. Como é possível observar, a entidade veículo é subordinada às outras, com exceção de proprietário, pois depende de uma série de chaves estrangeiras para que possa existir[Date 2004]. Além disso, um veículo pode ter uma série de proprietários ao longo do tempo, assim criou-se uma associação entre essas entidades.

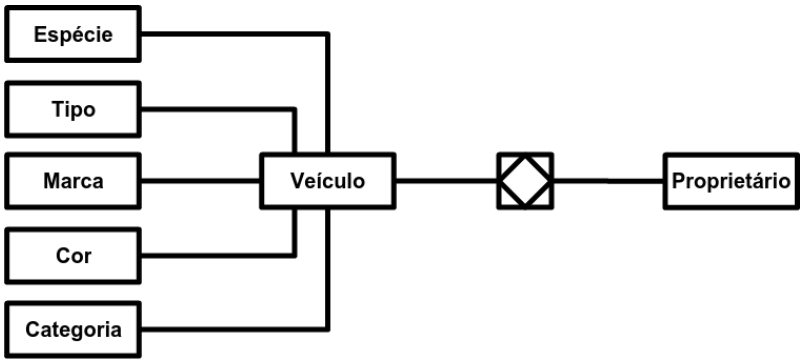

Figura 2. Modelagem resumida (simplificada) do banco de dados.

Após estudo do ambiente, foram realizadas diversas experimentações em baixa e larga escala, estudos matemáticos, modelagem do sistema, pesquisa de técnicas de conversão de arquivos e bibliográfica.

3. Soluções Propostas

A seguir são apresentadas duas propostas de inserção dos registros na nova base de dados. A primeira proposta possui uma abordagem convencional utilizada pelo framework Django e a segunda possui um caráter de otimização para a realidade da aplicação em questão.

Em ambas as propostas foram realizadas tarefas de seleção e pré-processamento de dados, aplicando-se um algoritmo Python, manipulado pelo framework Django, para a limpeza dos dados e definição dos conjuntos de dados consistentes e inconsistentes. Em outras palavras, esse algoritmo fez a leitura do `arquivo1` e classificou em dados consistentes aqueles registros que seguiam a codificação UTF-8 e que não possuíam duplicatas. Os demais registros foram mantidos no arquivo de inconsistências para serem tratados numa nova etapa do projeto.

3.1. Proposta 1: inserção convencional dos registros

A primeira proposta consiste na importação dos dados consitentes do `arquivo1` utilizando o método clássico de persistência de dados executadas pelo framework Django. Como mencionado, os dados passaram por um processo de seleção e pré-processamento,

como ilustrado pela Figura 3. Logo, à medida que as linhas do arquivo eram lidas classificadas como registros consistentes, chamava-se o método padrão de inserção em banco de dados utlizados pelo framework Django. Esses métodos são generalizados, assim, a consistência dos dados se dá através de uma busca completa por identificadores únicos de proprietário a cada inserção de veículo. Caso o proprietário não exista, um novo registro de proprietário é inserido, e posteriormente, sua chave primária é inserida na tabela associativa. Quando o CPF ou CNPJ já estiver cadastrado na tabela de proprietários, faz-se apenas sua associação com o veículo.

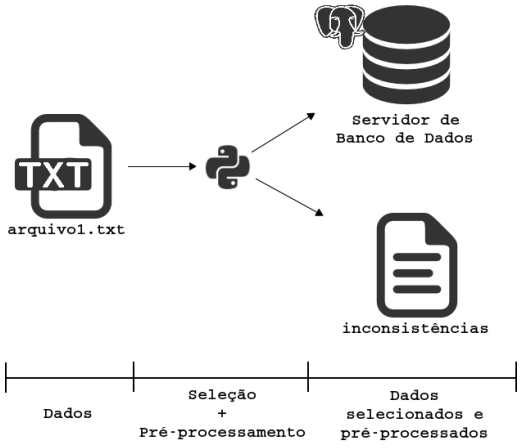

Figura 3. Processo de importação convencional da primeira proposta.

Foram realizados testes através da importação da base completa. Os resultados, apresentados na Seção 4, demonstraram que a adoção de uma nova proposta de importação era necessária.

3.2. Proposta 2: inserção otimizada dos registros

A segunda proposta trata da separação dos dados, atraés de uma função Python, em dois arquivos (`veiculos.csv` e `proprietarios.csv`) e sua posterior inserção no banco de dados, Através de outra função Python, conforme ilustrado na Figura 4.

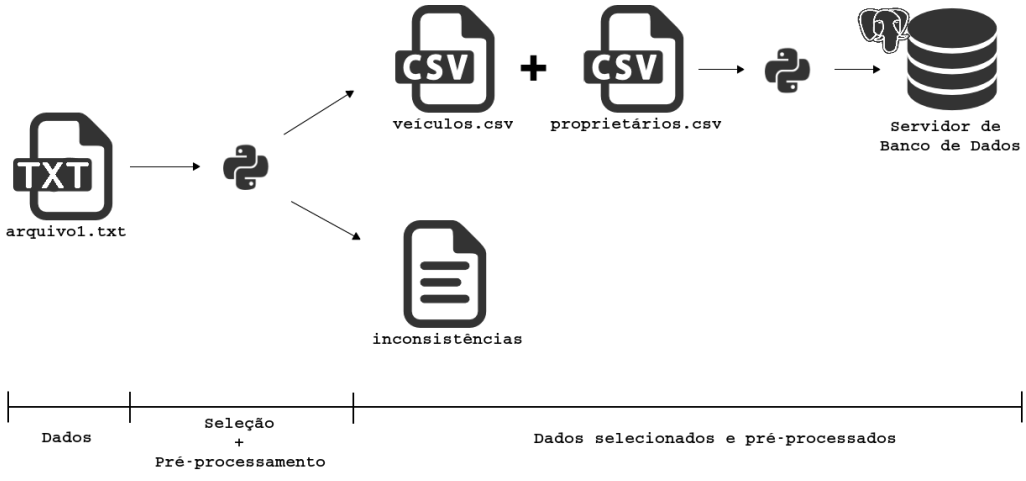

Figura 4. Processo de importação: separação em arquivos especialistas.

Para garantir a integridade dos dados, `proprietarios.csv` é percorrido em busca da chave primária (chassi ou CPF/CNPJ): caso repetições sejam encontradas para a mesma chave, apenas a primeira ocorrência é mantida.

Os veículos são inseridos através de um código SQL que não considera as características como chaves estrangeiras, mas como texto. Dessa forma, quando as outras tabelas são populadas, as chaves primárias são as mesmas que nos arquivos de texto puro, e a ligação pela chave estrangeira fica garantida. A seguir, a entidade associativa entre veículos e proprietários é construída, utilizando-se o chassi e o CPF/CNPJ contidos no arquivo `veiculos.csv`.

O código abaixo (Listing 1), trata do processo de importação de veículos para o banco de dados. A função foi resumida para aumentar a clareza em sua leitura.

Listing 1. Código que implementa função de inserção de registros no servidor de banco de dados.

```
def importa_veiculos(veiculos):
  erros_importa_veiculo = []
  cur = connection.cursor()

  try:
      cur.executemany(insert_veiculo, veiculos)
      connection.commit()
  except Exception as e:
      connection.rollback()

      for insert_item in veiculos:
          try:
              cur.executemany(insert_veiculo, [insert_item])
              connection.commit()
          except Exception as ex:
              connection.rollback()

              erros_importa_veiculo.append(
                  {'erro': 'insert_veiculo', 'ex': '%s;%s'
                      % (type(ex), ex), 'item': insert_item})
  cur.close()

  return erros_importa_veiculo
```

A função importa_veiculos recebe uma lista de veículos, provindos do arquivo `veiculos.csv`. Após o início de uma nova conexão com o banco de dados, ocorre a tentativa de efetivar o salvamento das informações. Caso ocorra algum erro, ele é inserido em uma lista de erros. Por fim, a função retorna uma lista de erros.

Como pode ser observado no código abaixo (Listing 2) os proprietários são vincunlados aos veículos, através da função vincula_veiculo_proprietario. Essa função recebe uma lista de veículos, e uma vez que a conexão com o banco de dados é concluída, tenta inserir os dados do proprietario e do veiculo como texto puro. Caso ocorra alguma falha

durante a inserção, o erro é inserido em uma lista de erros, que é por fim retornada.

Listing 2. Código que implementa a vinculação de veículos aos respectivos proprietários durante a inserção de registros no servidor de banco de dados.

```
def vincula_veiculo_proprietario(veiculo_proprietario_list):
  insert_veiculo_proprietario = '''INSERT INTO
  detransapp_veiculoproprietario(veiculo_id,proprietario_id,
  data) values (%s,%s,now())'''

  erros_importa_veiculo_proprietario = []
  cur = connection.cursor()

  try:
      cur.executemany(insert_veiculo_proprietario,
          veiculo_proprietario_list)
      connection.commit()
  except Exception as e:
      connection.rollback()

      for insert_item in veiculo_proprietario_list:
          try:
              cur.executemany(insert_veiculo_proprietario,
                  [insert_item])
              connection.commit()
          except Exception as ex:
              connection.rollback()

              erros_importa_veiculo_proprietario.append({
                    'erro': 'insert_veiculo_proprietario',
                    'ex': '%s;%s' % (type(ex), ex),
                    'item': insert_item})
  cur.close()

  return erros_importa_veiculo_proprietario
```

4. Resultados e Discussão

Na primeira proposta, a importação foi executada por cerca de 72 horas no servidor. Durante esse período, o processador utilizou 100% de sua capacidade. Estima-se que a importação do arquivo completo levaria cerca de 170 horas (7 dias). O gráfico da Figura 5 demonstra o tempo necessário para a inserção de veículos utilizando as técnica descritas. Como é possível observar, a medida que os registros são inseridos, o tempo aumenta de forma exponencial. Esse aumento ocorre devido às consultas feitas pelo banco de dados, sejam elas: busca por chaves primárias de veículos duplicados, busca pelas chaves estrangeiras, inserção de proprietários na respectiva tabela e a consequente consulta por chaves primárias repetidas.

Observa-se que o processo descrito se refere à importação tradicional dos dados,

utilizando-se dos pacotes nativos do framework, ou seja, além das operações tradicionais do banco de dados, conforme descrito por [Schatz et al. 2010], são realizadas operações de alto nível específicas do framework Django. Por definição, um framework é uma ferramenta de uso generalizado, um conjunto de funções genéricas que auxiliam no desenvolvimento [Pree 1995], de tal forma que uma aplicação específica, como esse trabalho, sofre com o baixo desempenho das operações em alto nível realizadas pelo framework.

O gráfico da Figura 5 ilustra, ainda, o processo de inserção de dados de veículos no servidor conforme descrito na segunda proposta. Assim, é possível perceber o ganho em desempenho, uma vez que a inserção de veículos necessita de apenas 2 horas, ou seja, quando a responsabilidade pela consistência dos dados é retirada do framework, e tratada através de código pela equipe, as operações de inserção têm melhor desempenho, devido ao fato de que a busca por chaves duplicadas é desconsiderada.

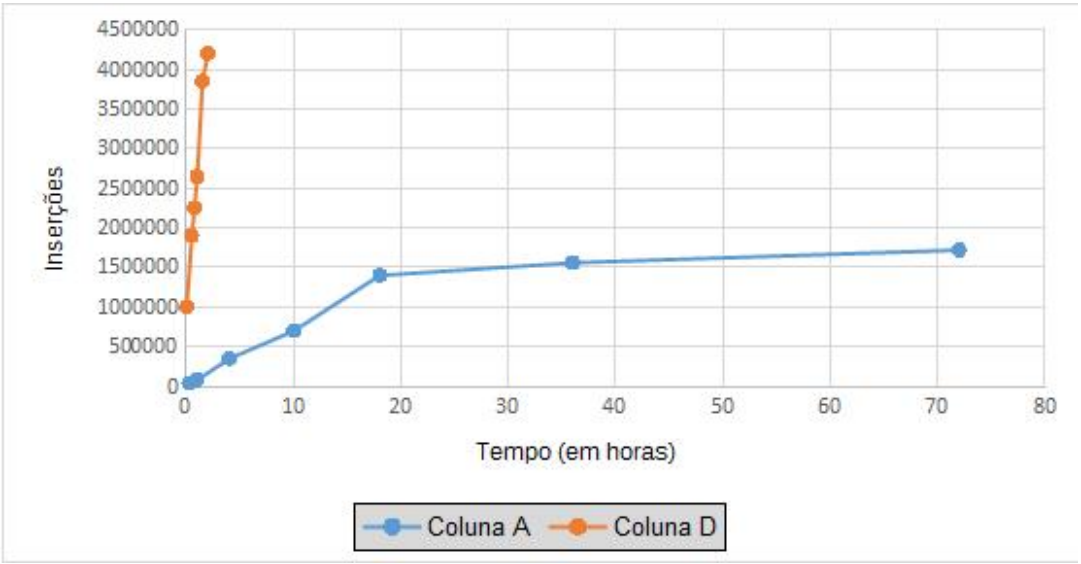

Figura 5. Gráfico de inserção de registros em relação ao tempo da estratégia inicial.

É importante ressaltar que os registros inconsistentes representam uma parcela de aproximadamente 5% dos dados, de tal forma que foram desconsiderados. Assim, eliminou-se o ruído que pode ser uma causa da lentidão do processo inicial de importação. Os dados foram mantidos para que, em uma nova etapa do projeto, sejam restaurados e o banco de dados mantenha a totalidade dos dados iniciais.

5. Considerações Finais

O objetivo principal desse trabalho foi apresentar uma solução de otimização de importação dos dados, oriundos de arquivos de texto, para um servidor de banco de dados relacional. Foram apresentadas duas propostas para a solução do problema. A primeira proposta tratou do inserção dos registros apenas utlizando o suporte oferecido pelo framework Django. A segunda proposta consistiu na separação dos dados do arquivo original, o `arquivo1`, em outros dois arquivos formatados, `veiculos.csv` e `proprietarios.csv`.

Os resultados mostraram que a segunda proposta se mostrou mais eficiente para a solução do problema, uma vez que a inserção de dados foi realizada em tempo menor. Além disso, as buscas por chaves, realizadas pelo banco de dados foram reduzidas, visto que os dados foram separados em novos arquivos específicos.

Com esse trabalho, demonstra-se que a mudança de estratégia pode ser a solução para problemas de baixo desempenho na importação de grandes volumes de dados. O tratamento dos dados e da integridade feito fora do banco de dados resolveu os problemas de desempenho. Sem a perda da generalidade, ao se considerar que a grande maioria dos dados utiliza a codificação UTF-8, aqueles aqueles que não estão nesse grupo foram desconsiderados.

Como parte dos dados devem ser mantidos em uma base de dados nos dispositivos móveis, os objetivos futuros incluem a utilização de índices no processo de popular essa base de dados. Para isso, será utilizado a arquitetura REST para estabelecer a comunicação entre os ambientes web e móvel descrita por [Rolim et al. 2014].

Além disso, pretende-se estudar novas estratégias de otimização utlizando outras tecnologias de persistência de dados, como por exemplo o NoSQL. Ainda, pretende-se verificar a influência da memória cache durante a inserção dos registros.

Referências

[Date 2004] Date, C. J. (2004). *Introdução a Sistemas de Banco de Dados*. Campus, 8 edition.

[Mota and et al. 2014] Mota, C. J. and et al. (2014). A experiência do ambiente da Fábrica de Software nas atividades de ensino do curso de Sistemas de Informação do IFC - Campus Araquari. *Anais do XXXIV Congresso da Sociedade Brasileira de Computação – CSBC 2014*, pages 1539 – 1548.

[Pree 1995] Pree, W. (1995). Design Patterns for Object-Oriented Software Developmen. *Addison Wesley*.

[Rolim et al. 2014] Rolim, V. B., Silva, M. R. d., Holderbaum, J., and Silva, E. d. (2014). A utilização da arquitetura REST para a comunicação entre diferentes plataformas. *Anais da VII Mostra Nacional de Iniciação Científica e Tecnológica Interdisciplinar - MICTI*.

[Schatz et al. 2010] Schatz, L. R., Viecelli, E., and Vigolo, V. (2010). PERSISTE: Framework para persistência de dados isolada à regra de negócios. *Congresso Sul Brasileiro de Computação*, 5.