Implementação de um gerenciador de tabelas como frontend para um banco de dados orientado a grafos

Matheus Leite Cruz

26/11/2018

Sumário

1 Problema

1.1 Apresentação do problema

Na definição do trabalho final, foi apresentada a seguinte ideia como projeto:

Criar uma interface de visualização unificada dos dados referentes a serie histórica dos indicadores nacionais de ciência tecnologia e inovação. Os dados estão disponíveis em diversas tabelas. A ideia é unificar todas as tabelas e outros dados relacionados em um único banco de dados local e permitir as ordenação e pesquisa dos dados por critérios, e se possível a representação gráfica dos dados onde for relevante.

Dado a definição acima, o programa precisa preencher os seguintes requisitos e realizar as seguintes funções:

- Ler tabelas do Excel no formato xls
- Criar alguma estrutura de dados para conter todas as tabelas lidas
- Gerar um banco de dados a partir da estrutura logica dos dados lidos
- Persistir dados
- Ordenar e pesquisar os dados lidos
- Inserir ou deletar dados
- Representar os dados lidos e/ou banco de dados visualmente

1.2 Programa desenvolvido

Para cumprir com todas especificações, foi desenvolvido um programa de gerenciamento de tabelas que implementa a visualização do banco de dados e suas estruturas por meio da biblioteca curses e funções para a interpretação de queries sobre o banco de dados.

O banco de dados do programa é representando por uma estrutura similar a uma arvore trie com tabelas como nodos, e as próprias tabelas podem ser representadas tanto quanto grafos quanto como matrizes, dependendo da necessidade da busca ou query.

É possível realizar diversos tipos de busca(detalhadas no guia de uso) sobre os dados e manipular esses dados em estruturas próprias dentro do programa("Containers") por meio das queries. Containers são objetos que podem representar vários tipos de estruturas logicas(detalhadas na implementação) como Células, Lista de Células, Tabelas, Dicionários ou Nodos.

Figura 1: Tela inicial do programa com a ajuda aberta descrevendo as queries possiveis

2 Implementação

2.1 Organização dos arquivos

O programa utilizou git para controle de versionamento, e pode ser encontrado online em https://github.com/mlcruz/SimpleGraphNoSQL

Organização dos arquivos:

```
raiz
├── fast -> Pasta com poucas tabelas para debugging
│   └── [tabelas]
├── tabelas_mcti
│   ├── fonte -> tabelas fonte
│   │   └── [tabelas]
│   └── formatada
│       └── 2
│           └── [tabelas] -> tabelas que começam com o indice 2
├── aux_lib.py -> biblioteca agregadora de funções
├── curseless.py -> biblioteca para testes sem a interface curses
├── db.py -> classe referente ao banco de dados
├── main.py -> entrada principal do programa
├── menus.py -> biblioteca contendo a interface curses e o interpretador de queries
├── Tabela.py -> Biblioteca contendo os tipos Table e Cell e seus respectivos metodos
└── Trie.py -> Classe referente a uma trie
```

2.2 dependencias

Todas as dependencias(tirando o curses para windows) podem ser instaladas utilizando o pip para python 3.6 64bits/windows

- sys -> Operações de sistema, utilizado para limpar tela do console e mudar limite de recursões
- curses -> Gerenciamento de tela. Instalada manualmente para windows a partir de ⟨https://www.lfd.uci.edu/~gohlke/pythonlibs/#curses⟩ [1]
- collections -> defaultdict para estrutura de dados
- dill -> serializa objetos [2]
- os -> gerencia chamadas ao sistema operacional
- easygui -> gerencia janelas e diálogos [3]
- curses.textpad -> gerencia entradas de texto no curses
- xlrd -> auxilia na leitura de tabelas do excel em formato xls [4]
- regex -> suporte para expressões regulares melhor que o da biblioteca padrão [5]
- unicodedata -> remocao de caracteres especiais de strings entre outros

2.3 Estruturas de dados

2.3.1 Cell

Classe que representa uma célula de uma tabela. Recebe um objeto do tipo sheet do xrld como entrada e uma posição X referente a linha e Y referente a célula. Cria um objeto do tipo célula classificando o tipo de célula e guardando dados.

Células são o tipo mais básico de dados compostos presentes em um banco de dados. Células podem ser representadas como folhas ou nodos(dependendo do seu tipo) de um grafo tabela ou células de uma tabela ordenadas por colunas e chaves de maneira posicional. Cada célula contem diversos atributos para representar as informações contidas na mesma.

Inicialmente células são inicializados sem os dados referentes a outras células da tabela, e pertencem a matriz(representada por um defaultdict(dict)) table.table_data[X][Ý] onde X e Y são as posições das células na tabela. Após a inicialização a tabela é varrida preenchendo as relações e tipos entre células dependendo de suas posições(por exemplo, uma key_row está sempre abaixo de uma raiz que está sempre na posição 0,0) e gerando o grafo que tem origem na célula(0,0) e representa uma tabela.

Dos atributos da célula, os seguintes são os mais importantes(ver fonte ou documentação autogerada para mais detalhes).

- cell.data − > Representa o dado contido na celula em questão
- cell.cell_type -> Representa o tipo da célula, definido pela sua posição na tabela em relação com outras celulas
  Tipos possiveis:

  Leaf -> Representa uma célula do excel contendo dados e mais nada. É relacionada a duas chaves, uma de linha(key) e outra de coluna(key_col)

  Key_Row -> Representa uma chave que ordena chaves de linha. Uma por tabela(Ex. ANO)

  Key -> Representa uma chave presente em uma Key_col. Ordena uma linha de leafs

  Super_Key -> Representa uma chave que é pai de outras key_col.

  Label -> Representa o nome da tabela, é a raiz da arvore e pai de todos os outros nodos

  Merge -> Representa uma célula que foi unida uma outra célula e não tem dados

  Blank ->Não faz parte da tabela, dado vazio que pode ser removido sem alterações

- cell.parent_node -> Representa o local da célula pai da célula atual.

  Key_Row é pai de todas as Key abaixo,

  Super_key é pai de todas as Key_col abaixo.

  Key é pai de todas as leafs em sua linha

  Key_col é pai de todas as leafs em sua coluna

  Label é a raiz, logo é pai de todas as células diretamente abaixo da raiz

- cell.child_nodes -> Lista contendo todos os nodos filhos de uma celula em questão. ver cell.parent_node para regras de atribuição
- cell.cell_name -> Nome da celula levando em conta seus pais. Representa relação pai-filho como >

Exemplo para Key(Filha de key_row que é filha de Label) onde

key = 2002;

key_row = ano;

label = "Teste Table"

Então:

cell.cell_name = "Teste Table>ano>2002"

2.3.2 Table

Classe que representa uma tabela. Contem um dicionario representando uma matriz de todas as suas células e algumas informações referentes a tabela como um todo. A classe tabela gerencia a inicialização e preenchimento de relações entre células de uma mesma tabela. Dos atributos da tabela, os seguintes são os mais importantes:

- table.table_label − > Nome da tabela, representado pela data da célula (0,0)
- table.table_data − > Matriz de celulas da tabela, raiz na posição table_data[0][0]
- table.key_cols − > Lista de key_cols da tabela
- table.key_row − > key_row da tabela
- table.keys − > Lista de keys da tabela

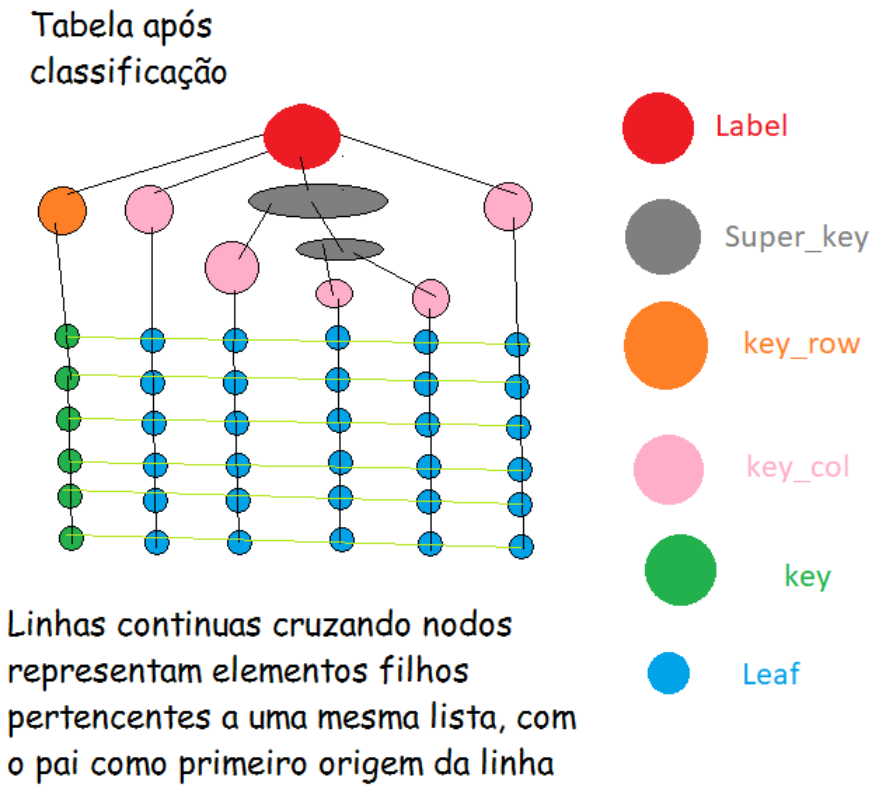
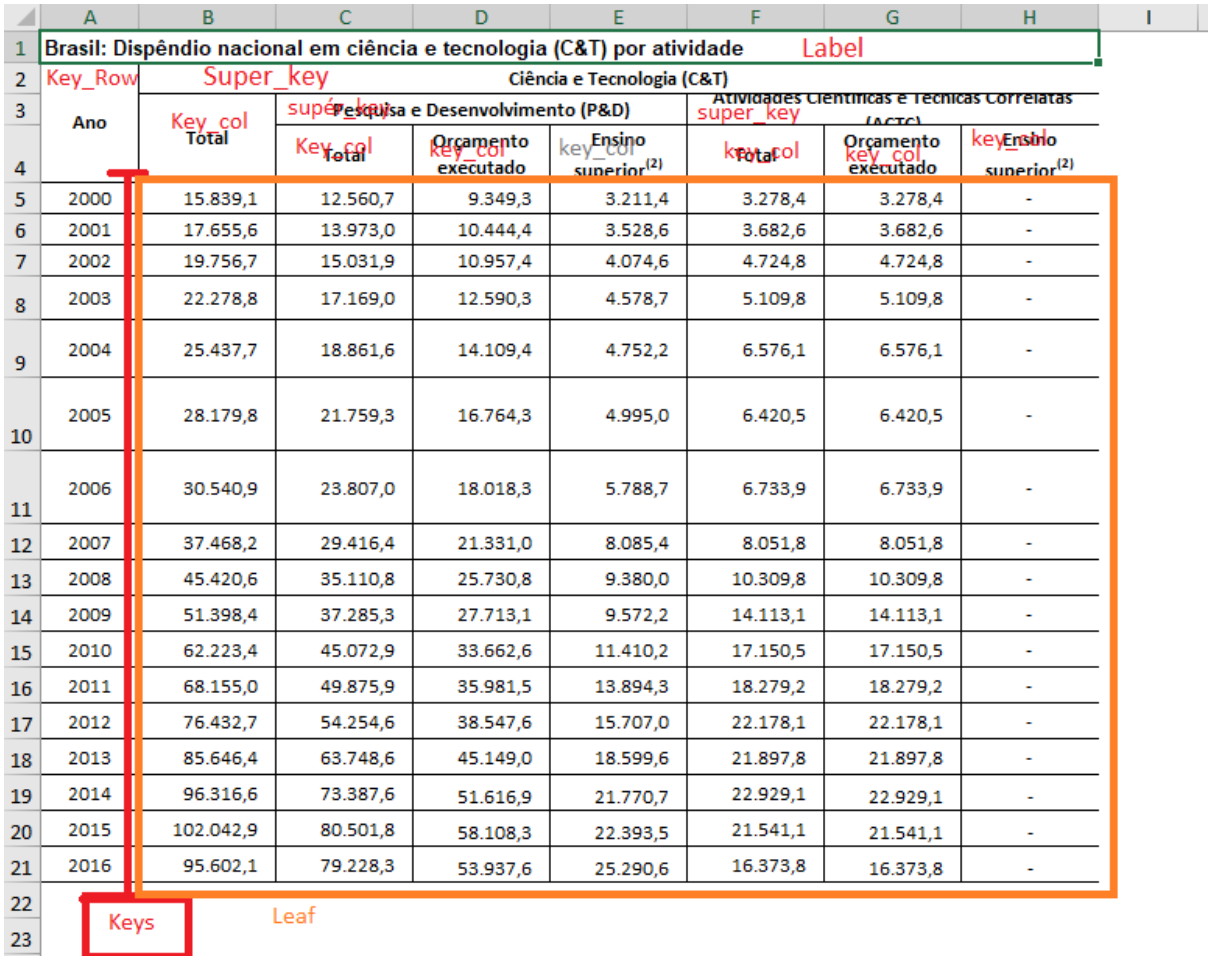


Figura 2: Tabela como grafo



	Ano	Total	Pesquisa e Desenvolvimento (P&D) Total	Pesquisa e Desenvolvimento (P&D) Orçamento executado	Pesquisa e Desenvolvimento (P&D) Ensino superior(2)	Atividades Científicas e Técnicas Correlatas (ACTC) Total	Atividades Científicas e Técnicas Correlatas (ACTC) Orçamento executado	Atividades Científicas e Técnicas Correlatas (ACTC) Ensino superior(2)
5	2000	15.839,1	12.560,7	9.349,3	3.211,4	3.278,4	3.278,4	-
6	2001	17.655,6	13.973,0	10.444,4	3.528,6	3.682,6	3.682,6	-
7	2002	19.756,7	15.031,9	10.957,4	4.074,6	4.724,8	4.724,8	-
8	2003	22.278,8	17.169,0	12.590,3	4.578,7	5.109,8	5.109,8	-
9	2004	25.437,7	18.861,6	14.109,4	4.752,2	6.576,1	6.576,1	-
10	2005	28.179,8	21.759,3	16.764,3	4.995,0	6.420,5	6.420,5	-
11	2006	30.540,9	23.807,0	18.018,3	5.788,7	6.733,9	6.733,9	-
12	2007	37.468,2	29.416,4	21.331,0	8.085,4	8.051,8	8.051,8	-
13	2008	45.420,6	35.110,8	25.730,8	9.380,0	10.309,8	10.309,8	-
14	2009	51.398,4	37.285,3	27.713,1	9.572,2	14.113,1	14.113,1	-
15	2010	62.223,4	45.072,9	33.662,6	11.410,2	17.150,5	17.150,5	-
16	2011	68.155,0	49.875,9	35.981,5	13.894,3	18.279,2	18.279,2	-
17	2012	76.432,7	54.254,6	38.547,6	15.707,0	22.178,1	22.178,1	-
18	2013	85.646,4	63.748,6	45.149,0	18.599,6	21.897,8	21.897,8	-
19	2014	96.316,6	73.387,6	51.616,9	21.770,7	22.929,1	22.929,1	-
20	2015	102.042,9	80.501,8	58.108,3	22.393,5	21.541,1	21.541,1	-
21	2016	95.602,1	79.228,3	53.937,6	25.290,6	16.373,8	16.373,8	-

Figura 3: Tipos de celula em uma tabela

2.3.3 Node

Representa um Nodo de uma Trie. É representando por uma objeto contendo os seguintes atributos

- nodo.child − > Defaultdict(dict) representando os filhos do nodo.

  Um defaultdict(dict) cria uma nova entrada no dicionario ao receber uma chave inexistente para consulta. A nova entrada é formada pelo par (key:value) onde key é a chave que não existia e value um object do tipo dict vazio. Logo, ao ser consultado e não existir automaticamente cria um novo proto-nodo

- nodo.parent − > pai do nodo
- nodo.chard − > atributo representando char que levou até o nodo atual
- nodo.data − > representa uma entrada de dados no nodo atual.

2.3.4 Trie

Representa uma Trie. Tries são a estrutura de dados principal do banco de dados, e contem vários tipos de dados diferentes do mesmo. Tries são formadas por Nodes, e a raiz de uma trie

é um nodo com chard = 0 e nenhum parent. Cada nodo de uma trie representa um caractere de alguma string, e cada vertice é representando por um caractere indicando o proximo nodo, e no ultimo nodo de uma string estão guardados os dados(sejam estes tabelas ou outros).

O grau dessa trie é maior que 26 pois ceita qualquer caractere como vértice que liga o próximo nodo.

A inserção em uma trie é feite por uma função insert(trie,string,dados). A função cria uma lista de caracteres baseado na string normalizada e insere recursivamente cada caractere na trie(criando um nodo se não existe caractere inserido na posição e tratando de não sobrescrever dados sobre strings já inseridas)por meio de acessos aos sucessivos aos defaultdics de cada nodo no caminho. Ao esvaziar a lista com os caracteres, insere os dados recebidos no campo data do nodo em questão.

Atributos de uma Trie:

- Trie.root − > Nodo representando a raiz da trie
- Trie.strings_dict − > Dicionario representando todos os dados de uma trie. É gerado pela função yield.strings que caminha recursivamente a trie e preenche o dicionario, que é usado como cache para acesso instantaneo
- Trie.strings_list − > Lista representando todos os dados de uma trie para accesso sequencial
- Trie.reverse − > Representa a trie revertida da trie atual, para busca por sufixo.

2.3.5 db

Representa um banco de dados. Um banco de dados contem diversas Tries, utilizadas para diversas consultas. Um banco de dados gerencia a reversão de suas tries e a geração de suas tries a partir da tabelas fonte ou do arquivo de persistência(por meio da biblioteca aux_lib).

Atributos de um banco de dados

- db.tables − > Representa a trie com todas as tables de um banco de dados.

  É gerado inserindo o par ¡label,Table¿ de cada tabela na trie.
- db.key_rows − > Representa a trie com todas as key_rows de um db.tables

  É gerada inserindo o nome(no formato label¿pai¿[n pais¿filho) de cada key e a lista de seus filhos como data em uma trie. Tem o objetivo de permitir a pesquisa de keys e key_rows
- db.key_cols − > Representa a trie com toda as key_cols de uma db.tables.

  É gerada inserindo ¡key_col.name,key_col.child_nodes¿ em uma trie. Tem como objetivo permitir a pesquisa por key_col

2.3.6 Container

Representa um tipo misto(Cell—List—Table—Nodo—dict—default_dict—default_dict(dict)) que guarda os dados das para consultas do usuário. Implementa funções de visualização dos objetos contidos dependendo do seu tipo no terminal(utilizando a biblioteca curses para funções adicionais). Serve como output ou input de uma query, mas o detalhamento de seu funcionamento foge um pouco do escopo do trabalho.

Atributos do Container:

- Container.data $->$ Representa dado salvo pelo container no seu tipo original(cell, table, etc..)
- Container.type $->$ Tipo do objeto contido pelo container
- Container.name $->$ Nome do objeto do container.

2.4 Leitura de tabelas

A leitura de tabelas do excel é realizada por varredura, utilizando a biblioteca xlrd para ler os dados de cada célula da tabela especificada e inicializar o objeto célula correspondente. É utilizada a função de retornar todos os items de um diretório do os, e então cada tabela do diretório selecionado é inicializada como objeto tabela e inserida na trie

2.5 Persistência

A persistência de dados é gerenciada por serialização sequencial simples do dicionario com todas as tables de uma Trie db.table de um banco de dados(db.table.strings_dict gerado por yield.strings, para ser exato) por meio da biblioteca dill. O objeto dicionario é serializado e salvo no disco como um arquivo binário, e pode ser deserializado pela mesma biblioteca.

O carregamento de um banco de dados envolver deserializar o objeto criado pelo dill e gerar todo o banco de dados de novo a partir da inserção das tabelas do dicionario(e respectivas labels). O processo é simplesmente inserir todas as tabelas em uma db.tables e gerar todas as tabelas dependentes desta de novo. É utilizado a biblioteca dill para facilitar a serialiação como objeto binário, mas como o objeto inicial é um par valor;chave seria facilmente possível serializar o mesmo de outra maneira (Ex:JSON)

2.6 Interface gráfica

A interface gráfica foi desenhada manualmente utilizando a biblioteca curses e loops. O processo foge do escopo do trabalho, mas é bem simples(mas trabalhoso)

2.7 Entrada e Saída do usuário

A entrada e saida do usuário é gerenciada por objetos Container contendo os dados recebidos. O processo de gerenciamento de contêineres é um simples dicionario de estados que salva ou recupera as entradas do usuário como <chave,Container> dependendo do contexto

2.8 Queries

A manipulação de dados do programa e suas funções é feito por queries entradas pelo usuário. O programa recebe a string da query, interpreta ela e faz alguma coisa com algum Container recebido(definido pelo usuário) e retorna para algum outro Container recebido(definido pelo usuário). Interpretar a query é um problema simples e foge do escopo do trabalho.

Praticamente toda a logica do programa foi pensada na utilização de queries pelo usuário e elas são responsáveis pela implementações dos critérios 3, 4 e 5 do trabalho. Uma query tem a seguinte estrutura command;value ou comand;trie!value onde comand é o comando a ser executado, value os dados passados pelo comando e se existir trie a trie onde deve ser realizada a operação

As funções definidas pelas queries e suas implementações são as seguintes. Cabe ao usuário escolher o tipo correto de container para as queries:

Queries de banco de dados(input_container não importa):

- pre;<trie>!<string>

  Realiza uma busca por prefixo, utilizando a string como prefixo, na trie especificada.

  Caminha até o nodo dado pela string na trie especificada, e realiza uma pesquisa em todos os nodos abaixo deste, retornando um dicionario de ¡label,tables¿ com uma entrada para cada nodo com dados. Retorna o dicionario no container de saída

- suf;<trie>!<string>

  Realiza uma busca por sufixo, utilizando a string como sufixo, na trie especificada.

  Caminha até o nodo dado pela string revertida na reversa da trie especificada, e realiza uma pesquisa em todos os nodos abaixo deste, retornando um dicionario de <label,tables> com uma entrada para cada nodo com dados. Retorna o dicionario no container de saída

- reg;<trie>!<string>

  Realiza uma busca por expressão regular, utilizando a string como expressão regular, na trie especificada. Caminha a trie inteira e coloca todos os dados em uma lista, e então varre a lista buscando matches na expressão regular recebida. Retorna um dicionario de <label,tables> no container de saida

- get_all;<trie>

  Retorna em um container o dicionario de tabelas da trie especificada.

- insert;<trie>

  insere container de entrada na trie especificada. Retorna dado inserido

- get;<trie>!<string>

  Recebe tabela do dicionario de tabelas da trie onde string == table_label

- yield;<trie>

  chama trie.yield_strings() para recarrega o dicionario de strings da trie.

- walkto;<trie>!<string>

  Caminha string na trie especificada e retorna o nodo no final da string no container de saida

- moonwalkto;<trie>!<string>

  Caminha string para a raiz no nodo atual e devolve a nodo final no container

- delete;trie!<string>

  Caminha até nodo na trie especificado e deleta dados. Retorna dado deletado no container de saída

Queries de tabela(precisa de uma tabela como container de entrada)

- key_col;<null | col_name>

  Retorna toda key_col(copia a lista de key_cols) da tabela se vazio ou todas as chaves da key_col especificada no container de saída.

- key_row;<null | row_name>

  Retorna a key_row da tabela se vazio ou todas as chaves de índice entre a e b no container de saída.

- insert_cell;<y,x>

  Insere Célula no container de entrada na posição y,x. Retorna um container com a célula inserida

- delete_cell;<y,x>

  Remove célula da tabela na posição y,x. Retorna célula removida no container de saída

- get_cells;<uy,ux,ly,lx>

  Retorna a lista de celulas entre as posições <uy,ux> e <ly,lx> no container de saída

- get_cell;<y,x,>

  Retorna a célula em y,x no container de saída

Queries de lista(precisa de uma lista como container de entrada)

- unpack;<null>

  retorna a lista contendo todas as células filho das células de uma lista de células no container de saída

- first;<null>

  retorna primeira célula da lista no container de saída

- rest;<null>

  retorna resto das células da lista no container de saída

- select;<index>

  retorna célula no índice especificado no container de saída

- remove;<index>

  remove célula no índice especificado e retorna no container de saída

Queries de célula(precisa de uma célula como container de entrada)

- edit<data>

  Edita a data de uma célula no container de entrada, retorna célula no container de saída

Queries de dicionario(precisa de um dicionario como container de entrada)

- pop<null>

  Remove um item do dicionario no container de entrada e retorna no container de saída

Queries de nodo(precisa de um nodo como container de entrada)

- get_all<null>

  Retorna no container de saída todos as tabelas embaixo de um nodo no container de entrada

2.9 Vantagens e desvantagens

Vantagens da implementação do modo descrito acima:

- Utilizar uma Trie como estrutura de dados para os dados sobre tables(e key_cols/rows) permite a pesquisa eficiente de informações por prefixo e sufixo, além de ser bem eficiente em termos gerais até na pesquisa sobre todos os nodos(como uma expressão regular)
- Alem dá Trie Utilizar dicionarios(hash tables) para o acesso imediato dos dados(quando não precisa pesquisar) permite uma eficiência ideal $\Theta(1)$ no acesso aos dados já pesquisados
- Representar a tabela como um grafo permite uma visualização completa da relação entre cada célula e sua tabela no banco de dados(por meio do 'nome completo' único de cada célula dada pela soma de seus dados e os dados de todos os seus nodos pais até a raiz) e representar cada célula individualmente em um dicionario permite acesso de qualquer célula em $\Theta(2)$
- Gerenciar as interações do usuário por meio de queries é facilmente extensível e permite muita flexibilidade(por exemplo, é possível tornar essa proto linguagem turing completa sem muitas extenções)
- Pesquisas eficientes em todas as tabelas por meio das Tries de key_col e key_row são simples de se realizar por meio de busca na string de nome com as operações já definidas anteriormente

Desvantagens:

- Implementar o modelo descrito acima foi muito trabalhoso, e o projeto ficou muito maior do que o esperado anteriormente(Cerca de 3000 linhas de código se incluído o arquivo tex de documentação).
- Por ficar maior do que o esperado o relatório do projeto se tornou muito extenso e mesmo assim o funcionamento do programa foi abordado superficialmente
- O uso de memoria ram pode estourar rápido ao salvar o objeto de banco de dados dada a inexistência de tail recursion optimization em python
- Dado o tamanho do projeto algumas funcionalidades não puderam ser implementadas. O tempo gasto implementando todas as queries foi menos de 10% do total gasto no trabalho, então muitas funcionalidades que poderiam ser facilmente implementadas não foram por falta de prazo

3 Guia de Uso

O uso do programa é simples: É possível gerar um banco de dados(por meio das opções exibidas no terminal) por meio de uma pasta de tabelas ou objeto salvo anteriormente, e após gerado o banco de dados é possível executar queries sobre containers (queries detalhadas acima ou ao pressionar o botão de ajuda no menu) e selecionar contaieners de saída e entrada para essas queries.

É possível também visualizar os dados dos containers ao pressionar a tecla referente a sua chave e salvar o banco de dados atual e limpar a area de exibição de tabelas. Todas as funcionalidades do programa estão divididas em 4 menus no total.

4 Considerações Finais

Acredito que o objeto do programa foi atingido, por mais que algumas coisas que seriam simples de implementar deixaram de ser implementadas por questão de tempo. O programa tem limitações em relação ao tipo de tabela que pode ser usado como fonte de dados(precisa que a label seja um merge que cobre toda a tabela, basicamente), então as tabelas precisam ser ligeiramente pre-formatadas antes de serem usadas como fonte de dados, além dos problemas de otimização de recursão em python.

No geral a maior dificuldade encontrada foi relacionada a escala do programa e o tempo necessário pra completar o que eu considerava como o mínimo de funcionalidades viável(e desenhar a interface gráfica)

Acredito que quase todo conteúdo da segunda área disciplina foi(tries, b+ trees, hastables) foi aplicada ou usado como modelo para alguma estrutura de dados utilizada, e a parte sobre bancos de dados NoSql relacionado a grafos foi a inspiração inicial para o projeto.

5 Referencias

Referências

1 GOHLKE, C. *Curses - Unofficial Windows Binaries*. Disponível em: ⟨https://www.lfd.uci.edu/~gohlke/pythonlibs/#curses⟩.

2 MMCKERNS. *dill*. Disponível em: ⟨https://pypi.org/project/dill/⟩.

3 ALEXANDER.ZAWADZKI et al. *Easygui*. Disponível em: ⟨https://pypi.org/project/easygui/⟩.

4 CHRISW; SJMACHIN. *Xlrd*. Disponível em: ⟨https://pypi.org/project/xlrd/⟩.

5 MRABARNETT. *Regex*. Disponível em: ⟨https://pypi.org/project/regex/⟩.

6 Apendice: Atributos das classes gerado pelo doxygen

SimpleGraphNoSql

Generated by Doxygen 1.8.14

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

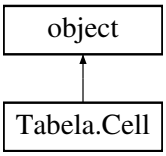
Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

Class Documentation

3.1 Tabela.Cell Class Reference

Inheritance diagram for Tabela.Cell:



Public Member Functions

- def **__init__** (self, sheet, X, Y)

Public Attributes

- data_boundary

  Inicializa celula, definindo atribuitos e o tipo da celula self.cell_type define o tipo da variavel: Tipos possiveis: Leaf -> Representa uma celula do excel contendo dados e mais nada.
- **blank_boundary**
- **originx**
- **originy**
- **sizex**
- **sizey**
- **x**
- **y**
- **bounds**
- **cell_type**
- **data**
- **table_label**
- **parent_node**
- **child_nodes**
- **key_row**
- **key_col**
- **cell_name**

3.1.1 Detailed Description

```
Classe que representa uma celula da tabela. Recebe um objeto do tipo sheet do xrld como entrada
    e uma posicao X referente a linha e Y referente a celula. Cria um objeto do tipo celula classificando o ti
```

3.1.2 Member Data Documentation

3.1.2.1 data_boundary

```
Tabela.Cell.data_boundary
```

Inicializa celula, definindo atribuitos e o tipo da celula self.cell_type define o tipo da variavel: Tipos possiveis: Leaf -> Representa uma celula do excel contendo dados e mais nada.

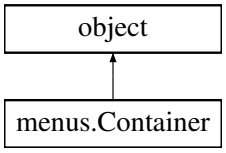
É relacionada a duas chaves, uma de linha(key) e outra de coluna(key_col) : Key_Row -> Representa uma chave que ordena outras chaves(exemplo: ANO). Uma por tabela Key_Col ->Representa uma chave que dá sentido a uma coluna. Podem existir varias por tabela Key -> Representa uma chave presente em uma Key_col Super_Key -> Representa uma chave que é pai de outras chaves ou de uma chave coluna. Label -> Representa o nome da tabela, é a raiz da arvore e pai de todos os outros nodos Merge -> Representa uma celula que foi unida uma outra celula e não tem dados Blank ->Não faz parte da tabela, dado vazio que pode ser removido sem alterações

The documentation for this class was generated from the following file:

- C:/Users/PC/source/repos/Trabalho_final_cpd/SimpleGraphNoSQL/Tabela.py

3.2 menus.Container Class Reference

Inheritance diagram for menus.Container:



Public Member Functions

- def **__init__** (self, data)

Public Attributes

- **raw_type**
- **name**
- **data**
- **type**

3.2.1 Detailed Description

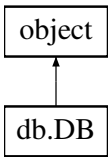
```
Classe representando um objeto que pode ser um Nodo ou uma Tabela ou uma Lista de celulas ou uma Celula. data
```

The documentation for this class was generated from the following file:

- C:/Users/PC/source/repos/Trabalho_final_cpd/SimpleGraphNoSQL/menus.py

3.3 db.DB Class Reference

Inheritance diagram for db.DB:



Public Member Functions

- def __init__ (self, trie)

Public Attributes

- **tables**
- **key_rows**
- **key_cols**
- **super_key**

3.3.1 Detailed Description

```
Cria banco de dados a partir de uma trie de tabelas
```

3.3.2 Constructor & Destructor Documentation

3.3.2.1 __init__()

```
def db.DB.__init__ (
              self,
              trie )
```

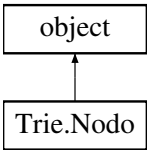
```
Inicialize banco de dados com uma trie recebida
```

The documentation for this class was generated from the following file:

- C:/Users/PC/source/repos/Trabalho_final_cpd/SimpleGraphNoSQL/db.py

3.4 Trie.Nodo Class Reference

Inheritance diagram for Trie.Nodo:



Public Member Functions

- def __init__ (self, char_data, data)

Public Attributes

- **child**
- **parent**
- **chard**
- **data**

3.4.1 Detailed Description

```
Representa um nodo de uma trie
```

3.4.2 Constructor & Destructor Documentation

3.4.2.1 __init__()

```
def Trie.Nodo.__init__ (
              self,
              char_data,
              data )
```

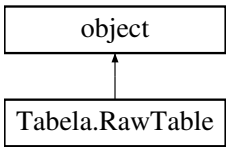
```
Inicializa nodos com dados. char_data é o caractere, e data são os dados ligados aquele nodo. Representar nenh
```

The documentation for this class was generated from the following file:

- C:/Users/PC/source/repos/Trabalho_final_cpd/SimpleGraphNoSQL/Trie.py

3.5 Tabela.RawTable Class Reference

Inheritance diagram for Tabela.RawTable:



Public Member Functions

- def __init__ (self, loc)

Public Attributes

- **loc_source**
- **raw_table**
- **raw_sheet**
- **table_label**
- **table_data**
- **bound_x**
- **bound_y**
- **key_cols**
- **key_row**
- **keys**

3.5.1 Detailed Description

```
Classe que representa uma tabela do excel na memoria. Guarda mais dados do que o necessario e não consegue ser
```

3.5.2 Constructor & Destructor Documentation

3.5.2.1 __init__()

```
def Tabela.RawTable.__init__ (
              self,
              loc )
```

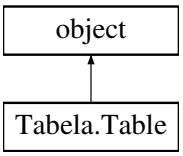
```
Inicializa uma tabela a partir de um local loc
```

The documentation for this class was generated from the following file:

- C:/Users/PC/source/repos/Trabalho_final_cpd/SimpleGraphNoSQL/Tabela.py

3.6 Tabela.Table Class Reference

Inheritance diagram for Tabela.Table:



Public Member Functions

- def __init__ (self, raw_table)

Public Attributes

- **loc_source**
- **table_label**
- **table_data**
- **bound_x**
- **bound_y**
- **key_cols**
- **key_row**
- **keys**

3.6.1 Detailed Description

```
Classe que representa uma tabela logica. Recebema uma Raw_Table e remove os campos desnecessarios Pode ser pic
```

3.6.2 Constructor & Destructor Documentation

3.6.2.1 __init__()

```
def Tabela.Table.__init__ (
              self,
              raw_table )
```

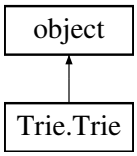
```
Inicializa Tabela com dados da raw table
```

The documentation for this class was generated from the following file:

- C:/Users/PC/source/repos/Trabalho_final_cpd/SimpleGraphNoSQL/Tabela.py

3.7 Trie.Trie Class Reference

Inheritance diagram for Trie.Trie:



Public Member Functions

- def __init__ (self)
- def **yield_strings** (self, trie)

Public Attributes

- **root**
- **strings_dict**
- **strings_list**
- **reverse**

3.7.1 Detailed Description

```
Implementa uma Trie para pesquisa no nome de tabelas de maneira eficiente
```

3.7.2 Constructor & Destructor Documentation

3.7.2.1 __init__()

```
def Trie.Trie.__init__ (
              self )
```

```
Inicializa raiz da trie como um defaultdict com uma factory de dicionarios
```

The documentation for this class was generated from the following file:

- C:/Users/PC/source/repos/Trabalho_final_cpd/SimpleGraphNoSQL/Trie.py

Index