ARTHUR ADABO DE CAMARGO
PEDRO ALVARES EGGERS

DESENVOLVIMENTO DE UM APLICATIVO DE PROPRIEDADES DE COMPOSTOS COM AROMA

Departamento de Engenharia Química

São Paulo, 2020

ARTHUR ADABO DE CAMARGO
PEDRO ALVARES EGGERS

DESENVOLVIMENTO DE UM APLICATIVO DE PROPRIEDADES DE COMPOSTOS COM AROMA

Trabalho de Conclusão de Curso apresentada à Escola Politécnica da Universidade de São Paulo como requisito parcial para Graduação no Curso de Engenharia Química

Orientador:
Prof. Dr. Moisés Teles dos Santos

São Paulo
2020

Autorizo a entrega dessa versão da Monografia sob responsabilidade única dos autores e de seus orientadores.
São Paulo, 18 de abril de 2020

Assinatura do orientador: _____
(Prof. Dr. Moisés Teles)

Assinatura do autor: _____
(Arthur Adabo de Camargo)

Assinatura do autor: _____
(Pedro Alvares Eggers)

de Camargo, Arthur Adabo; Eggers, Pedro Alvares
DESENVOLVIMENTO DE UM APLICATIVO DE PROPRIEDADES DE COMPOSTOS COM AROMA / A. A. de Camargo, P. A. Eggers – São Paulo, 2020.
65 p.

Trabalho de Conclusão de Curso – Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Química

1. Aroma de compostos 2. Interface gráfica em Python 3. Banco de dados I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Química II.t.

RESUMO

A industria de cosméticos, assim como muitas outras, depende de aromas artificiais para que seus produtos sejam mais atrativos ao cliente. Para obtenção desses aromas, é necessário conhecimento de propriedades fisico-químicas de moléculas – como a pressão de vapor – para que elas se encaixem nos requerimentos dos produtos. Assim, pode-se utilizar o método de Computer-Aided Molecular Design (CAMD), que pretende obter uma molécula a partir de certas propriedades, como, no caso, o aroma.

O uso de um banco de dados que permita fácil acesso a propriedades das moléculas é essencial para que especialistas ou até programas automatizados possam formular novos compostos com aromas agradáveis e que sigam as necessidades do produto, como baixa toxicidade, certo ponto de ebulição ou solubilidade específica.

Neste projeto, foram utilizadas as linguagens *SQLite* e *Python* para construir um banco de dados correlacionando informações de diferentes fontes e uma interface que seja capaz de acessar um banco de dados que contenha informações sobre compostos, tentando identificar compostos com propriedades desejadas e calcular valores julgados relevantes e que possam ser úteis para o desenvolvimento de novos produtos pela indústria de cosméticos bem como por outras em que aromas sejam de particular interesse.

DEVELOPMENT OF AN AROMA COMPOUNDS PROPERTIES SOFTWARE

ABSTRACT

The cosmetics industry, as well as many others, relies on artificial aroma to make their products more attractive to their customers. To obtain these aromas , it is necessary to know the physico-chemical properties of molecules - such as vapor pressure - to fit the requirements of the products. Thus, the so-called Computer-Aided Molecular Design (CAMD method, which seeks to obtain a molecule given certain properties, such as the aroma, may be used.

The use of a database that allows easy access to molecule properties is essential for specialists or even automated programs to formulate new compounds with pleasant aromas that satisfies product needs, such as low toxicity, a certain boiling point or a specific solubility.

In this project, the SQLite and Python languages were used to build a database that contains correlating compound properties obtained from different sources and an interface that can access it, trying to identify characteristics and associate them with odors, as well as calculate relevant values which may be useful for the development of new products by the cosmetics industry as well as others in which aromas are of a particular interest.

LISTA DE FIGURAS

LISTA DE TABELAS

NOMENCLATURA E SÍMBOLOS

Siglas

CAMD	*Computer Aided Molecular Design*
GUI	*Graphic User Interface*
SMILES	*Simplified Molecular Input Line Entry Specification*
IUPAC	*International Union of Pure and Applied Chemistry*
csv	*Comma Separated Values* (formato de arquivo)

SUMÁRIO

1 INTRODUÇÃO

Na indústria química, de maneira geral, os compostos utilizados em determinado processo são conhecidos, sendo exploradas as suas propriedades em dadas condições. Por exemplo, em um processo de separação de uma mistura composta por tolueno e benzeno, é importante saber quais propriedades – como densidade, calor específico etc. – estas substâncias possuem nas temperaturas e pressões utilizadas.

O *Computer Aided Molecular Design* é o processo inverso: sabendo as propriedades desejadas em certas condições, busca-se um composto químico que as apresente. Este processo envolve tanto buscas simples em bancos de dados como análise de modelos moleculares para prever características relevantes da molécula, como seus grupos funcionais.

Na formulação de um perfume ou outros aromatizantes, muitas vezes sabe-se qual será o seu aroma (floral, amadeirado, doce) antes de ser definida qual será sua composição. No passo seguinte, especialistas em aroma, cujo conhecimento do ramo é essencialmente empírico, selecionam substâncias que atendam à necessidade do consumidor.

Nesse sentido, o uso de CAMD poderia, com base em informações do aroma, da pressão de vapor e de outras propriedades relevantes para a fragrância, fornecer combinações possíveis de compostos de maneira objetiva, eficiente e rápida.

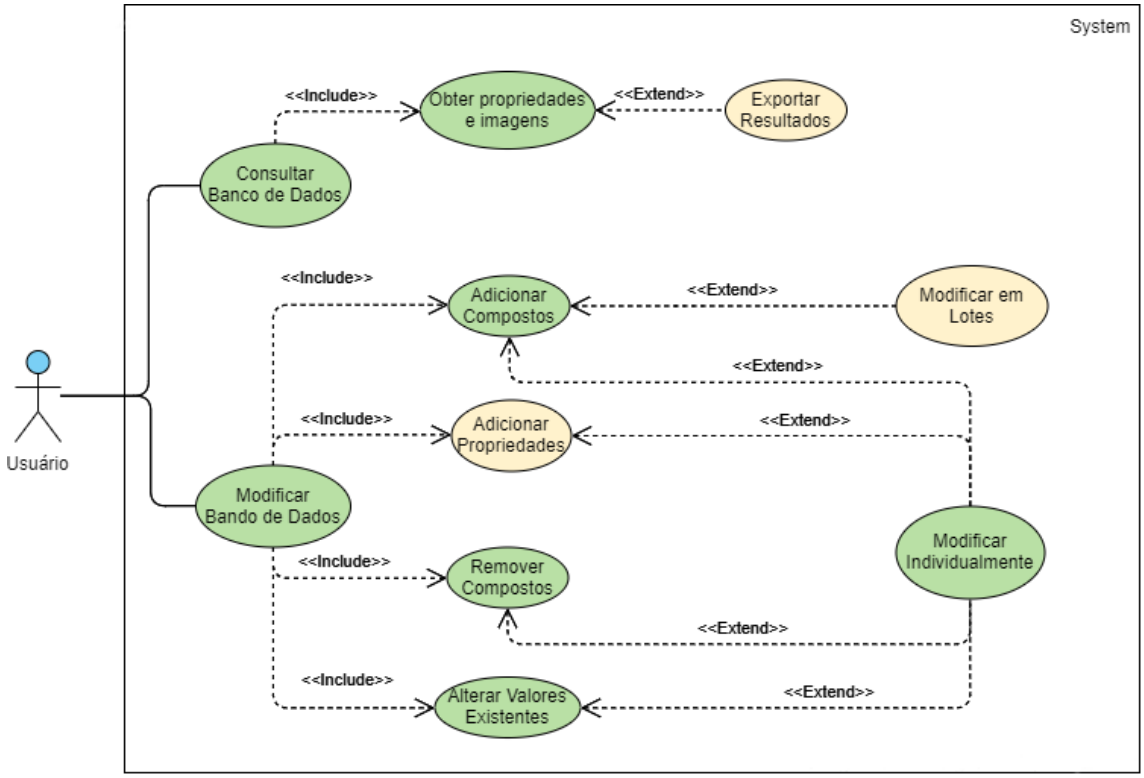
É importante ressaltar que o CAMD não tem como objetivo substituir o trabalho dos especialistas em aroma, que adicionam um componente pessoal e subjetivo aos perfumes, mas sim em auxiliá-los a obter mais combinações de substâncias que gerariam as características desejadas no produto em um tempo menor e com um custo mais baixo, direcionando os experimentos à formulações mais promissoras.

O perfumista David Apel, há 39 anos no ramo e tendo desenvolvido perfumes para marcas como Avon, Nina Ricci e Gap, afirma que aplicações de CAMD utilizando inteligência artificial – como a *Philyra*, da IBM – "dão a chance de encontrar fórmulas de perfumes que eu nunca teria pensado sozinho" (OSTERATH, 2019).

O presente trabalho tem como objetivo descrever a criação de um banco de dados contendo informações previamente não unificadas sobre propriedades e aromas de compostos, de um programa básico para auxiliar na escolha destes com base em suas propriedades bem como os métodos utilizados para cria-los. Inicialmente, será elaborado um banco de dados das substâncias cujo aroma seja relevante à indústria, escrito em SQLite, que será acessível via uma interface de usuário escrita em *Python*.

No escopo do trabalho também se inclui a funcionalidade do programa, o qual deve ser fácil e intuitivo de usar, tanto para usuários inexperientes quanto avançados. Uma descrição dos casos de uso atuais e futuros do programa está presente no modelo UML (Unified Modeling Language) na figura 1.

Figura 1 - Diagrama UML dos casos de uso



Fonte: Camargo e Eggers (2020).

Note que os elementos em verde são as funcionalidade que de fato foram inclusas no programa na data do término deste trabalho. Em amarelo, estão presentes as funcionalidades que este grupo de trabalho acredita serem interessantes para o escopo do projeto, mas ainda não foram incluídas. A descrição destas e de sua importância está descrita na seção “Trabalhos Futuros” deste relatório.

O resultado do trabalho em questão poderá ser ampliado futuramente para de fato usar modelos preditivos e *Machine Learning,* identificando moléculas com propriedades escolhidas pelo usuário.

1.1 Estrutura da apresentação do Projeto

O desenvolvimento do projeto está dividido da seguinte maneira:

Capítulo 2: descreve a metodologia do projeto, como o programa, sua aplicação e suas estruturas.

Capítulo 3: descreve as ferramentas e *softwares* utilizados.

Capítulo 4: apresenta o passo-a-passo da realização do projeto, bem como o algoritmo do programa criado.

Capítulo 5: apresenta os resultados do trabalho e exemplos de uso

Capítulo 6: apresenta sugestões para trabalhos futuros. .

Capítulo 7: apresenta as conclusões sobre o trabalho e o programa criado.

Apêndice: reúne o código dos três arquivos criados em *Python*.

2 DESCRIÇÃO DOS OBJETOS DO ESTUDO

2.1 Importância dos aromas para a indústria

A percepção sobre o aroma é um dos principais fatores de compra de produtos cosméticos. Estudos realizados na Europa (França, Alemanha, Itália, Espanha e Reino Unido) e nos Estados Unidos mostram que o produto ter uma fragrância que o cliente acha agradável é o primeiro fator de decisão para produtos de banho e o segundo para produtos desodorantes (IN-COSMETICS HAMBURG, 2014).

É importante ressaltar que, neste caso, o aroma está em segundo lugar na Espanha, Reino Unido e nos Estados Unidos, sendo priorizada uma proteção de longa duração

nos dois primeiros e a marca do produto no terceiro. Nos outros países citados, o aroma é ligeiramente mais valorizado do que a proteção.

Ademais, é importante saber que cada país terá uma preferência de aroma ao se decidir aonde determinado produto será veiculado. Por exemplo, um produto com aroma floral terá maior chance de sucesso em países como Brasil e Japão do que no Canadá e na Índia, visto que, de maneira geral, habitantes destes países preferem aromas frescos (IN-COSMETICS HAMBURG, 2014).

No Brasil, o investimento em aromas é particularmente interessante, uma vez que 85% dos possíveis clientes estariam dispostos a pagar mais em produtos com fragrâncias especialmente atrativas (IN-COSMETICS HAMBURG, 2014).

O componente “agradável” dos aromas, embora componha uma parcela considerável da sua valorização pelos clientes, não é o único que pode ser aproveitado na composição de produtos cosméticos. A composição aromática destes pode invocar memórias e sentimentos no cliente que criam uma conexão entre ele e a marca.

Além disso, o aroma de produtos é associado imediatamente à sua qualidade. Em estudos realizados por Fitzgerald e Swati em 2008 e Cox em 1969 (CLARK, 2009), concluiu-se que meias-calças com leves aromas agradáveis eram consideradas de melhor qualidade do que suas contrapartes sem aroma.

Desta forma, observa-se que produtos com aromas atrativos são mais rentáveis por serem associados a sensações agradáveis, memórias individuas e produtos de alta qualidade. Na indústria de cosméticos, estas características são ainda mais decisivas devido à finalidade dos produtos.

2.2 Propriedades importantes para aromas

Para a criação de fragrâncias, é necessário conhecer algumas propriedades físico-químicas dos compostos com aroma que serão utilizados (OETTERER, 2015):

- Aroma
- Taxa de vaporização
- Solubilidade
- Sensibilidade à fotossíntese
- Instabilidade (propensão a reações)
- Volatilidade

Destas propriedades, foram incluídas no banco de dados criado neste trabalho o aroma, a solubilidade e a pressão de vapor (associada à taxa de vaporização e volatilidade). As demais não foram encontradas.

2.3 Compostos com aroma

Como os compostos com aromas são de interesse da indústria, convém que outras propriedades suas sejam avaliadas em sua escolha. No trabalho em questão, as seguintes propriedades e caracterísicas foram incluídas no banco de dados:

- Identificador SMILES
- Aroma
- Nome
- Fórmula molecular
- Ponto de Ebulição
- Ponto de Fusão
- Ponto de Flash
- Solubilidade em Diferentes Compostos
- Pressão de Vapor
- Densidade
- Densidade de Vapor
- pKa

2.4 Banco de dados de compostos

De forma a armazenar informações relevantes sobre compostos com aromas, bem como permitir a pesquisa de maneira fácil e rápida, foi elaborado um banco de dados para tais substâncias.

O banco de dados é constituído basicamente de uma grande tabela na qual cada linha representa um composto único e as colunas representam as propriedades dos mesmos.

Vale ressaltar que muitas das propriedades citadas no tópico 2.1 variam conforme as condições do meio, bem como com solventes. Por isso, o banco de dados foi criado de modo a incluir, quando possível, mais de uma condição para o valor da propriedade.

2.5 Bancos de dados existentes

Devido à importância das propriedades de compostos com aroma para a indústria, já existem alguns softwares voltados para esta temática. Dentre eles, destacam-se:

- AromaDB (KUMAR et. al., 2018)[1];
- FooDB, contendo informações e diversas propriedades sobretudo para compostos da indústria de alimentos;
- FlavorNet (ACREE; ARN, 2004), contendo apenas o aroma de diversos compostos, sendo mais reduzido do que os anteriores;
- *The Good Scents Information System*, um banco de dados extenso e voltado para a indústria de cosméticos e de alimentos.

Ademais, o grupo se baseou em trabalhos preexistentes relativos à criação de bancos de dados de compostos e interfaces gráficas para óleos vegetais (TELES DOS SANTOS et. al, 2014) e produtos naturais (VALLI, M. et. al, 2013). Isso mostra que há uma demanda em diversos segmentos para uma aplicação como a apresentada neste trabalho.

As aplicações referenciadas contam com a exposição de diversas propriedades de compostos que correspondem à entrada do usuário, mas apenas a interface para os óleos vegetais apresenta funcionalidade de adição de compostos ao banco.

A interface apresentado no programa voltado a produtos naturais, bem como alguns dos bancos de dados disponíveis na internet, também conta com a exposição da fórmula estrutural dos compostos.

O projeto apresentado nesta monografia busca ampliar a aplicação dos bancos de dados para além do *browser* com uma *GUI* simples e leve e mesclar algumas de suas funcionalidades relevantes e úteis, focando seu uso nas indústrias em que aroma sejam relevantes. Além disso, há a possibilidade futura de se utilizar as extensas

[1] Até a publicação deste trabalho, o AromaDB não estava acessível, mas o artigo sobre seu desenvolvimento sim.

bibliotecas de estatística disponíveis em *Python* para estender o projeto de um banco de dados para um programa capaz de auxiliar com o *CAMD*.

2.6 Interface de Usuário

As interfaces de usuário, também conhecidas como *GUI*, são programas que facilitam a interação do usuário com códigos que executam certas funções.

A interface escrita em *Python* é constituída de uma janela contendo três seções:

2.6.1 Seção de Entrada de Dados

Esta seção está localizada na seção superior da janela e contém um menu *dropdown* para pesquisa de aromas e mais 11 caixas de texto para as demais propriedades apresentadas na seção 2.1. Além disso, conta com dois botões: “Adicionar”, que insere no banco de dados o composto escrito nas caixas de texto pelo usuário; e “Pesquisar”, que retorna correspondências no banco de dados à entrada do usuário nas caixas de texto.

2.6.2 Seção de Hyperlinks

Localizada na porção inferior esquerda da janela, essa seção contém *hyperlinks* das correspondências encontradas no banco de dados. Os *hyperlinks* contém a fórmula molecular do composto e seu nome.

Clicando-as, o usuário terá o resultado das demais propriedades bem como a imagem da estrutura da molécula – caso tal arquivo esteja presente na pasta “images” – na seção descrita no tópico a seguir.

2.6.3 Seção de Resultados

Localizada na porção inferior direita, exibe a imagem do composto selecionado junto de suas propriedades encontradas no banco de dados.

Exemplos da *GUI* e de entradas e saídas do programa estão apresentados na seção 5 deste documento.

3 REVISÃO DAS TECNOLOGIAS

Nesta seção, serão descritas as tecnologias utilizadas para elaborar o projeto, focando em suas características técnicas.

3.1 Banco de dados de compostos

A linguagem escolhida para o banco de dados em questão foi o *SQLite* devido ao fato desta ser simples, altamente funcional, rápida, gratuita e no domínio público, compatível com várias plataformas e com capacidade de armazenamento suficientemente grande para a aplicação neste trabalho. Como os bancos de dados em *SQLite* suportam até 140 TB de armazenamento, não haveria qualquer dificuldade técnica em armazenar um número grande de compostos em passos futuros do trabalho (SQLITE CONSORTIUM).

No momento da conclusão deste trabalho, haviam 234 compostos no banco de dados, ocupando apenas 100 kB de espaço. Nota-se, portanto, que o projeto é completamente escalável, não havendo uma limitação real à ampliação do banco de dados. Teoricamente, supondo um crescimento linear do espaço em disco ocupado pelo banco de dados com o número de linhas, poderiam ser armazenados 327,6 bilhões de compostos, tornando-o essencialmente ilimitado.

Além disso, o *SQLite* é a linguagem de banco de dados mais utilizada no mundo, estando presente em todos os aparelhos celulares com *Android* e *iOS*, todos os computadores com *Windows 10*, todos os *Macs*, nos navegadores *Chrome, Firefox e Safari* e em outras aplicações.

3.2 *GUI*, interação com o banco de dados e tratamento de dados

Para realizar as partes do programa relativas à *GUI*; interações com o banco de dados – criação, atualização de tabelas, bem como consultas às mesmas – e tratamento de dados, a linguagem escolhida foi *Python*.

Os motivos dessa escolha estão no fato do *Python* ser uma linguagem poderosa e versátil, contendo diversos módulos nativos ou não para as mais variadas aplicações.

Além disso, o número de aplicações baseadas em *Python* e de pessoas com conhecimento na linguagem têm crescido fortemente – 456% desde 2018 (BHAWANI, 2019) –, tornando mais provável que algum usuário do programa conheça a forma com que ele foi escrito.

Os módulos do *Python* utilizados na elaboração do programa estão apresentados com suas respectivas funções na tabela 1 a seguir:

Tabela 1 - Lista dos módulos do *Python* utilizados

Módulo	Função	Arquivo
Tkinter	Elementos visuais, como a janela do programa	*app.py*
PIL (Python Imaging Library)	Leitura de imagens de compostos no formato .png	*app.py*
os	Interação com elementos do sistema operacional (local de arquivos etc.)	*app.py, query.py, import_pubchem.py*
sqlite3	Interpretação e interação com *SQLite3*	*query.py*
csv	Leitura de tabelas em *.csv*	*query.py, import_pubchem.py, get_properties.py*
pubchempy	*API*[2] para obter dados de moléculas do *PubChem*	*import_pubchem.py*
xlsxwriter	Escrita em arquivos *.xlsx* do Excel	*import_pubchem.py*
requests	Solicitações HTTP[3]	*get_properties.py*
selenium	Controles de Browser (webdriver)	*get_properties.py*
time	Contagem de tempo e espera	*get_properties.py*

Fonte: Camargo e Eggers (2020).

[2] Acrônimo de "Application Programming Interface", utilizada para acesso a informações de um aplicativo baseado na Web
[3] Método de acesso a informações contidas na Internet

4 DESCRIÇÃO DO PROCESSO

Neste capítulo, serão explicados os programas utilizados para pré-processamento de dados e para consultas ao banco de dados de substâncias criado. Foram criados três programas, cada um para realizar um objetivo relacionado ao trabalho. As etapas realizadas foram as seguintes:

4.1 Tabulação de dados: SMILES e aroma

Com base em dados conhecidos sobre o aroma de compostos (KORICHI, 2008) e suas respectivas notações em SMILES, inicialmente organizou-se uma tabela com estes dados no *Excel*, sendo ela convertida ao formato *.csv*. Esta parte foi feita manualmente devido à dificuldade de coletar de maneira automática os dados do artigo citado. A notação SMILES (“simplified molecular-input line-entry system”, ou “sistema simplificado de entradas de moléculas em linha”) é uma forma de se representar uma molécula em apenas uma linha de texto seguindo o algoritmo desta notação.

Os aromas presentes em Korichi et. al. que foram incluídos no programa e poderão ser utilizados como parâmetro de pesquisa são:

- Anis
- Bálsamo
- Frutado
- Azedo
- Cânfora
- Mel
- Baunilha
- Doce
- Rosa

O formato *.csv* foi escolhido devido à facilidade de se trabalhar com o mesmo em *Python*, utilizando o módulo homônimo nesta linguagem para fazê-lo. Seria possível armazenar os dados em formatos *.xls* ou *.xlsx*, mas utilizando outro módulo no *Python* para trata-los, como o *Pandas*, por exemplo.

4.2 Importação de dados do *PubChem* via API

Com a tabela correlacionando a notação SMILES das moléculas com seus odores, o próximo passo foi obter mais dados – como nome IUPAC e fórmula química – sobre estas moléculas em bancos de dados confiáveis. No caso, foi escolhido o *PubChem* devido à sua extensão e confiabilidade, uma vez que ele é mantido pela *U.S. National Library of Medicine*[4]

Levando em conta a quantidade de compostos cujas informações deveriam ser buscadas, o preenchimento manual destes valores foi avaliado como inviável. Para automatizar esse processo, foi escrito um pequeno programa (*import_pubchem.py*) em *Python* utilizando o módulo *pubchempy*, disponibilizado pelo próprio *PubChem* (SWAIN, 2017).

Esta etapa começa com a execução da função *import_from_pubchem.* Nela, os dados presentes no arquivo criado no tópico 4.1 contendo os pares (SMILES, aromas) é carregado em uma lista. Isso é feito pela função *read_csv* presente no código.

Em seguida, a função *import_from_pubchem* cria um objeto[5] *xlsxwriter.Workbook* que representa o arquivo do *Excel* com informações de alimentação ao banco de dados, adicionando a ele uma planilha denominada *“results”.*

Para cada composto da lista criada pela função *read_csv*, o programa:

1. Pesquisa o seu nome e fórmula molecular no *PubChem* através da API, armazenando os resultados em uma lista através do método[6] *get_compounds()* da biblioteca *pubchempy*

2. Faz o download da imagem associada ao composto em questão através do método *download()* do *pubchempy*

3. Escreve os dados obtidos no arquivo *.xlsx* supracitado utilizando o método *write()* da biblioteca *xlsxwriter*.

[4] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4702940/, acesso em 09 abr. 2020
[5] Membro de uma classe
[6] Função associada exclusivamente a uma classe

Essa etapa consiste no pré-processamento dos dados que, embora tenha levado cerca de 10 minutos para ser concluída, é extremamente mais rápida do que o processo feito manualmente. Além disso, vale ressaltar que a API também permite o *download* de imagens das estruturas do compostos, que serão mostradas ao usuário.

Todavia, a obtenção de mais dados sobre as moléculas, como a densidade, pKa e solubilidade, levou mais tempo para ser feita devido ao fato de estas informações não estarem acessíveis via API.

Para contornar este problema com a API, foi criado um programa que agrega estas propriedades diretamente do site através do uso da biblioteca do python *requests*. A partir da análise das *URLs* de uma página de informação sobre um composto, é possível perceber que elas são formadas com 2 variáveis: o ID do composto atribuído pelo próprio PubChem como um número e qual propriedade é desejada.

Por exemplo, para a solubilidade do composto de ID 6616 (“Camphene”), tem-se o seguinte endereço:

https://pubchem.ncbi.nlm.nih.gov/compound/6616#section=Solubility&fullscreen=true

O programa *get_properties.py* trabalha em três etapas:

1. Realização do primeiro *request*[7] que retorna o ID do composto baseado numa pesquisa pelo seu SMILES, seguido por uma iteração numa lista de propriedades que foram julgadas importantes, sendo que, para cada propriedade da lista, é feito um *request* para a *URL*[8] formatada com o ID do composoto e a propriedade desejada.

2. Realização de uma busca no código-fonte da página para encontrar a informação utilizando elementos das linguagens *HTML* e *CSS* que pudessem identifica-la. No caso do PubChem, a informação era encontrada dentro de um *div*[9] cuja classe tem nome ‘section-content-item’.

[7] Requisição HTTP, explicada na nota de rodapé 3
[8] Acrônimo de “Uniform Resource Locator”, o endereço de algum site
[9] Elemento divisor da linguagem HTML

3. Escrever em um documento de formato CSV as informações encontradas.

Porém, nem todos os compostos no *PubChem* continham dados sobre estas propriedades, tornando o banco de dados incompleto em alguns pontos. É importante ressaltar que, mesmo que esse processo tenha levado algumas horas para ser feito, ele ainda representou um aumento na velocidade do projeto. Além disso, proporcionou ao grupo um importante aprendizado em *web-scraping*[10].

4.3 Criação do banco de dados

Após a obtenção dos dados relativos aos compostos de interesse, foi criado o banco de dados utilizando a linguagem *SQLite3*, feito também automaticamente através de um programa (*query.py*) em *Python*, utilizando o módulo *sqlite3* desta linguagem.

Embora alguns dos valores das colunas tenham sido encontrados facilmente, outros não foram, deixando o banco de dados incompleto em alguns pontos. Estes valores ainda estão sendo pesquisados em outras fontes. A tabela 2 a seguir explicita quantos compostos no banco de dados não possuem determinada propriedade:

Tabela 2 - Número de Compostos sem Determinadas Propriedades no Banco de Dados

Propriedade	Nº de Compostos
Ponto de Ebulição	101
Ponto de Fusão	121
Ponto de Flash	168
Solubilidade	126
Pressão de Vapor	147
Densidade	160
Densidade de Vapor	192
pKa	229

[10] Forma de se ler o código HTML de uma página da Web e obter informações buscadas de maneira programática.

O programa *query.py* contém funções que auxiliam na criação do banco de dados, da tabela que contém as propriedades dos compostos, na leitura destas e na subsequente inserção ao banco. São elas:

4.3.1 Função *create_db()*

A função *create_db()* tem a finalidade de criar o banco de dados de compostos caso ele ainda não exista. Inicialmente, é criada a conexão entre *Python* e banco de dados via a função *create_connection()*. Caso não haja erro – ou seja, o programa conseguiu estabelecer uma conexão –, há a execução da função *create_table()*.

No entanto, o novo banco de dados está vazio após a conclusão desta etapa, sendo necessário populá-lo. Isto é feito utilizando a função *update_db().*

4.3.2 Função *create_connection()*

A função *create_connection()* serve para criar a conexão entre *Python* e o banco de dados do SQLite. Ela recebe o caminho do banco de dados no computador do usuário e, através de uma estrutura *try...*except[11], caso não haja erros, retorna um objeto *sqlite3.Connection.*

4.3.3 Função *create_table()*

A função *create_table()* tem como finalidade criar uma tabela no banco de dados do SQLite. Ele recebe o objeto *sqlite3.Connection* e uma *String*[12] contendo a query de criação de tabela a ser realizada. Nesta função, há uma estrutura *try... except* que tenta criar um objeto *sqlite3.Cursor*[13] para executar a query dada como argumento da função. Caso não haja erros, a tabela de compostos será criada através do método *execute()* do objeto *sqlite3.Cursor*.

4.3.4 Função *update_db()*

A função *update_db()* recebe uma lista de valores, cria um objeto *sqlite3.Connection* e tenta – utilizando uma estrutura *try...except* – atualizar o banco de dados. Isto é feito criando um objeto *sqlite3.Cursor* e executando dois de seus métodos:

---

[11] Estrutura de programação na qual o programa tenta executar algum comando e, no caso de alguma exceção (por exemplo, um erro), realiza outra ação.
[12] Tipo de dado na computação que indica o formato de texto
[13] Objeto utilizado para realizar os comandos do SQLite

1. *executemany()*: utilizado para adicionar vários valores no banco de dados ao mesmo tempo, recebendo a query a ser feita em formato de *String* e uma lista de valores a serem adicionados.
2. *commit():* utilizado para confirmar e salvar as mudanças feitas.

Para gerar a lista de entrada desta função, foi utilizada a função *read_molecule_csv()*, muito similar à *read_csv()* presente no arquivo *import_pubchem*. A diferença é que a função *read_molecule_csv()* retorna uma lista de *tuples* para facilitar seu uso no método *executemany()*.

4.3.5 Função *read_molecule_csv()*

A função *read_molecule_csv()* abre o arquivo *.csv* contendo a informação sobre a notação SMILES extraído de Korichi et. al., cria um objeto *csv.reader* e o converte para uma lista de *tuples*, retornando-a.

Note que o método *fetchall()* retorna um "*tuple*[14] de *tuples*", sendo necessário convertê-lo a uma lista para atendar à finalidade da função.

4.3.6 Função *get_data()*

A função *get_data()* é utilizada nas buscas realizadas no banco de dados pelo usuário. Este método recebe 12 *Strings* como argumentos, uma para cada campo exposto no item 2.1 deste relatório. O método monta uma *query*[15] ao banco de dados, cria uma conexão e um objeto *sqlite3.Cursor* e obtém os resultados através do uso de dois métodos:

1. *execute()*: utilizado para executar a consulta ao banco de dados.
2. *fetchall():* utilizado buscar todos os resultados para a consulta.

Por fim, o método converte o resultado da *query* em uma lista e a retorna.

[14] Tipo de dado no qual pode-se armazenar informações de maneira listada e imutável
[15] Termo para "consulta" ao banco de dados.

4.3.7 Função *get_odours()*

A função *get_odours()* tem como finalidade consultar o banco de dados e obter uma lista em ordem alfabética dos aromas nele presentes. Esta lista é retornada pela função para popular o menu *dropdown* de aromas na interface de usuário.

Na seção 5 deste documento, estão expostas imagens exemplificando a estrutura do banco de dados.

4.3.8 Função *delete_compounds()*

A função *delete_compounds()* tem como finalidade remover um composto selecionado pelo usuário na *GUI* do banco de dados, recebendo como argumento este nome. A função tenta criar uma conexão com o banco de dados e criar um cursor para executar uma *query* DELETE. Em seguida, salva a mudanças e fecha a conexão.

Caso não haja erro em algum passo deste processo, a função retorna Verdadeiro. Caso contrário, retorna Falso. Desta maneira, é possível mostrar na *GUI* ao usuário se o processo de remoção foi feito corretamente.

4.4 Criação da interface gráfica do usuário (GUI)

De forma a tornar a busca ao banco de dados mais amigável a usuários, foi criada uma *GUI (Graphic User Interface)* também em *Python*, utilizando o módulo *tkinter* (SHIPMAN, 2013), cujo programa tem nome *app.py*.

O funcionamento deste programa depende da criação de uma classe[16] – chamada de **App** por representar a janela aberta ao usuário quando o arquivo é executado – com diversos elementos, a qual será descrita mais adiante neste tópico. Além desta classe, criou-se também uma para conter subdivisões da janela com barras de rolagem, denominada **ScrollFrame**, e outra para objetos *tk.Label* com quebras corretas de texto, denominada **WrappingLabel**. Ambas as classes serão descritas em detalhe mais adiante neste documento.

[16] Estrutura que, além de armazenar informações, pode conter métodos relacionados a seu funcionamento e interagir com outros objetos.

4.4.1 Execução do arquivo *app.py*

No início do programa *app.py*, há a criação de uma lista de aromas, os quais serão expostos na janela de interface. Em seguida, as classes **App** e **ScrollFrame** são declaradas.

Após a declaração, o cria-se uma instância da classe Tk sob o nome de "*root*", que será a janela do programa, mas não se inicializa o objeto, uma vez que é necessário configurar a janela primeiro.

Em seguida, criam-se dois objetos Menu:

- menubar: representando a barra de menu atrelada a root
- help_menu: representando o botão dropdown do menu com opões de ajuda ao usuário, adicionado com o método add_cascade da classe Menu.

No menu de ajuda, há três botões: um para ajuda sobre pesquisa de compostos, um para ajuda sobre a atualização do banco de dados e outro para informações sobre o programa. Todos os botões estão associados a funções simples que mostram uma caixa de diálogo – um objeto *tkinter.messagebox* – com informações relevantes. Os botões e suas respectivas funções foram configurados utilizando o método *add_command()* da classe Menu.

Terminando a criação de menus, o programa define a geometria da janela em 800x600 pixels e fixa este como o seu tamanho mínimo, evitando a desconfiguração dos widgets[17] no caso de alterações pelo usuário. Estas etapas são feitas utilizando os métodos *geometry()* e *minsize()* do objeto root.

Por fim, o programa cria o objeto da classe **App** e a mantém aberta através do método mainloop().

[17] Elementos gráficos da janela, como botões, campos de entrada, menus etc.

4.4.2 Classe *App*

A classe ***App*** descrita neste documento consiste na interface de usuário em si específica para a finalidade de pesquisa de compostos no banco de dados e sua atualização. Ela contém os seguinte métodos:

4.4.2.1 Método *__init__*

O método *__init__*, conforme citado anteriormente, é executado quando uma instância da classe é inicializada. Neste caso, o método recebe um objeto “pai” ou “âncora” – no caso, o objeto *root* criado anteriormente – e cria um objeto *tk.Frame* da biblioteca TKinter ancorado a ele.

Este objeto serve para organizar subdivisões da janela e facilitar posicionamento de objetos adicionais. A função deste *Frame* será conter três outros *Frames:* um para os campos e botões de pesquisa, um para os resultados e outro para as imagens da fórmula estrutural dos compostos. Estes *Frames* serão criados através dos métodos *create_search_frame(), create_results_frame() e create_image_frame()*, descritos adiante.

Ademais, o método recebe os argumentos **args* e ***kwargs*. Na convenção de escrita de *Python,* estes representam:

- **args*: uma lista dos argumentos passados, podendo iterar sobre os mesmos em ordem;
- ***kwargs*: um dicionário de argumentos que podem ser buscados pelo nome.

Como o objeto *tk.Frame* tem um número considerável de parâmetros, ao declarar os argumentos **args* e ***kwargs*, o programa deixa a cargo do usuário fornecer aquelas que ele deseja alterar, aumentando a possibilidade de personalização.

Por fim, há o estabelecimento de duas propriedades do objeto da classe **App.**

- *parent:* relativo a qual é o elemento “pai” do objeto;

- *results_button:* uma lista de *hyperlinks*[18] que será atualizada com os resultados de cada pesquisa, permitindo mostra-los para cada composto com um clique.

4.4.2.2 Método *search()*

Método para facilitar a leitura do código da classe. Ele simplesmente executa os métodos *submit()* e *update_results_frame()* sucessivamente.

4.4.2.3 Método *add_compound()*

O método *add_compound()* executa um passo de checagem, pedindo a confirmação do usuário sobre a adição do composto ao banco de dados através de uma caixa de mensagem. Esse passo é necessário para evitar a adição de compostos de maneira errada mas, caso o usuário ainda adicione uma espécie química ao banco sem desejar, poderá excluí-la depois com o método *delete_compound()*.

Em seguida, executa função *update_db()* do arquivo *query.py* fornecendo como argumento uma lista contendo um *tuple* contendo as informações de cada uma das barras de pesquisa da interface de usuário. Desta maneira, é possível adicionar um composto escolhido ao banco de dados para futuras consultas.

4.4.2.4 Método *create_search_frame()*

O método *create_search_frame* cria um objeto *Frame* com *widgets* à entrada do usuário e outro relacionado aos botões de pesquisa e adição de compostos. Inicialmente, estabelece-se a propriedade *search_frame* como um objeto *Frame* associado ao argumento do método. Este *Frame* servirá de âncora para os objetos adicionados a seguir.

Para cada um dos tópicos apresentados no item “Compostos com aroma” da seção 2 deste relatório – excetuando-se o aroma, o qual terá uma lista de possíveis valores – , são criadas barras de pesquisa usando o método *_create_search_bar*. Cada barra é associada a uma propriedade da classe ***App***, sendo seu nome “<*VALOR*>*_search_bar”*, em que “<VALOR>” diz respeito às possíveis propriedades citadas na seção 2.

[18] Texto que retorna algo quando clicado. No caso, retorna a visualização das informações do composto.

Para o aroma, cria-se um objeto *tk.Label* para identificar a que se refere a lista dropdown criada utilizando o método *create_dropbox*.

Após a adição destes *widgets*, são criados dois objetos *tk.Button:*

- *submit_button*: para pesquisar compostos no banco de dados
- *add_button*: para adicionar compostos ao banco de dados

Por fim, associa-se a tecla *Enter* ao método *submit()* de maneira a facilitar a busca de dados.

4.4.2.5 Método *_create_search_bar()*

O método *_create_search_bar()* recebe um objeto que servirá de âncora, coordenadas relativas X e Y da janela, uma largura relativa e uma altura absoluta para a barra de pesquisa, retornando um objeto *tk.Entry* relativo a uma propriedade pesquisável.

O método também é responsável por criar um objeto *tk.Label* para identificar ao usuário a qual propriedade a barra de pesquisa criada se refere. Este objeto é colocado nas coordenadas relativas X e Y fornecidas como argumento do método, enquanto a barra de pesquisa é posta ligeiramente abaixo desta posição.

Note que o método tem nome iniciado com "_". Na convenção do *Python*, isso indica que ele é um método privado e não deve ser importado pelo comando "import".

4.4.2.6 Método *create_dropbox()*

O método *create_dropbox()* recebe um objeto "âncora" e uma lista de valores a serem exibidos como opções, retornando um objeto *tk.OptionMenu*. Neste caso, a colocação do objeto na janela é feita fora deste método.

4.4.2.7 Método *create_results_frame()*

O método *create_results_frame* cria o objeto *ScrollFrame* – cujas características são descritas adiante neste documento – e o associa à propriedade *results_frame*. Em seguida, vinculam-se os atos de entrar e sair do *Frame* com o mouse aos métodos *_frame_entered()* e *_frame_left()*, cujo funcionamento será descrito adiante.

O método então realiza uma pesquisa de todos os compostos do banco de dados com o método *submit()* e atualiza o *results_frame* com os seus resultados.

4.4.2.8 **Método *create_image_frame()***

O método *create_image_frame* cria o objeto Scroll*Frame* com a imagem do composto selecionado pelo usuário após a busca. Este *Frame* é associado à propriedade *image_frame* do ***App*.** Há, igual ao que ocorre no método *create_results_frame()*, o vínculo da entrada e saída do mouse no *ScrollFrame* aos métodos *_frame_entered()* e *_frame_left()*.

Em seguida, cria-se um objeto *tk.Label* ao qual a imagem será configurada e há sua associação à propriedade *image_panel*. O painel da imagem é ancorado ao *image_frame*.

Com o método *update_image_frame()*, o painel da imagem passa a conter, inicialmente, a imagem relativa ao primeiro composto na lista de resultados da consulta ao banco de dados, feita automaticamente ao se executar o programa.

Vale ressaltar também que são criados objetos da classe *WrappingLabel* para cada propriedade exibida, de maneira que o texto do resultado para um composto, caso seja maior do que a largura do *Frame*, seja quebrado de maneira correta.

4.4.2.9 **Método *create_popup_menu()***

O método *create_popup_menu()* é responsável pela criação de um objeto *tk.Menu* associado ao clique com o botão direito do mouse. Desta maneira, ao clicar um dos *hyperlinks* de compostos no *Frame* de resultados na parte inferior esquerda da janela, o usuário terá a opção de remover o composto do banco de dados. Selecionando o “Delete”, é executado o método *delete_compound()*.

4.4.2.10 **Método *update_image_frame()***

O método *update_image()* recebe o local no computador da imagem de um composto, cria um objeto *tk.PhotoImage* com ela e a adiciona ao *image_panel*. Além disso, são adicionados os valores para as propriedades do composto selecionado pelo usuário no *Frame* abaixo da imagem.

4.4.2.11 Método *submit()*

O método *submit()* cria a propriedade *displayed_values*, que corresponde a uma lista de resultados da consulta ao banco de dados. Esta lista é criada através da função *get_data()* do arquivo *query.py*.

Para fornecer os argumentos à função *get_data()*, o programa utiliza o método *get()* para cada barra de pesquisa criada.

4.4.2.12 Método *delete_compound()*

O método *delete_compound()* é responsável por executar a função *delete_compound()* do arquivo *query.py*, removendo o composto do banco de dados.

4.4.2.13 Método *_handle_result_button_click()*

O método em questão basicamente executa o método *update_image_frame()* com base no evento de um clique a um botão exibido. Desta forma, é possível saber em qual dos botões o usuário clicou e, assim, exibir as informações corretas.

4.4.2.14 Método *update_results_frame()*

O método *update_results_frame()* cria um objeto *tk.Label* para cada linha da lista da propriedade *displayed_values*. Estes objetos representam os *hyperlinks* nos quais o usuário pode clicar para que o programa mostre as propriedades para o composto selecionado e sua imagem.

Para que isso seja possível, cada *hyperlink* é associado ao método *update_image_frame* e estabelece-se que o cursor do mouse do usuário seja um cursor de mão, indicando que o *hyperlink* pode ser clicado de fato.

4.4.2.15 Método *_frame_entered()*

O método *_frame_entered()* essencialmente verifica se o usuário entrou em um determinado *Frame* com o mouse. Desta maneira, o programa associa o botão de rolagem do mouse à barra de rolagem do *Frame* em questão através do método *_handle_scroll().*

4.4.2.16 Método *_frame_left()*

O método *_frame_left()* verifica se o usuário saiu de determinado *Frame* e desassocia o botão de rolagem do mouse a ele.

4.4.2.17 Método *_handle_scroll()*

O método *_handle_scroll()* torna possível a rolagem de um *Frame* com base na rolagem do mouse.

4.4.3 Classe *WrappingLabel*

A classe ***WrappingLabel****,* utilizada na criação de objetos no método *create_image_frame()*, consiste na associação de um objeto *tk.Label* a uma função que configura a extensão da quebra do texto nele presente à largura do *Frame* da imagem. Desta forma, o texto é exibido da maneira correta, sem quebras em segmentos errados do texto mesmo com a alteração no tamanho da janela.

Essa classe contém apenas o método *__init__*, criando um objeto *tk.Label* e utilizando o método *bind()* para associá-lo à função de configuração do texto com a quebra correta.

4.4.4 Classe *ScrollFrame*

A classe ***ScrollFrame*****,** utilizada na criação de objetos nos métodos *create_search_frame()* e *create_image_frame()* consiste na associação de um *Frame* com um objeto *ScrollBar*. Desta maneira, caso haja mais linhas em determinado conjunto de dados exibidos do que o possível de ser exibido na janela, o usuário pode utilizar a barra de rolagem para navegar pelas informações.

A classe ***ScrollFrame*** conta com os seguintes métodos:

4.4.4.1 Método *__init__*

Na inicialização de uma instância da classe ***ScrollFrame***, ela irá herdar, através da função *super()* do *Python*, as propriedades e métodos do *Frame* dado como argumento. Em seguida, há a criação de um objeto *tk.Canvas;* de um *Frame* dentro deste e de uma barra de rolagem no *Canvas*.

Por fim, vincula-se o evento de alterar o tamanho do *Frame* ou do *Canvas* aos métodos *onFrameConfigure()* e *onCanvasConfigure()*, descritos a seguir.

4.4.4.2 **Método *onFrameConfigure()***

O método *onFrameConfigure()* essencialmente reinicia a região da barra de rolagem para conter a região interna a ela, evitando desconfiguração do layout desta área. Isso é feito com a execução do método *configure()* do objeto *Canvas*.

4.4.4.3 **Método *onCanvasConfigure()***

O método *onCanvasConfigure()* realiza a mesma ação do *onFrameConfigure*, mas reiniciando o *Canvas,* corrigindo suas dimensões.

5 RESULTADOS

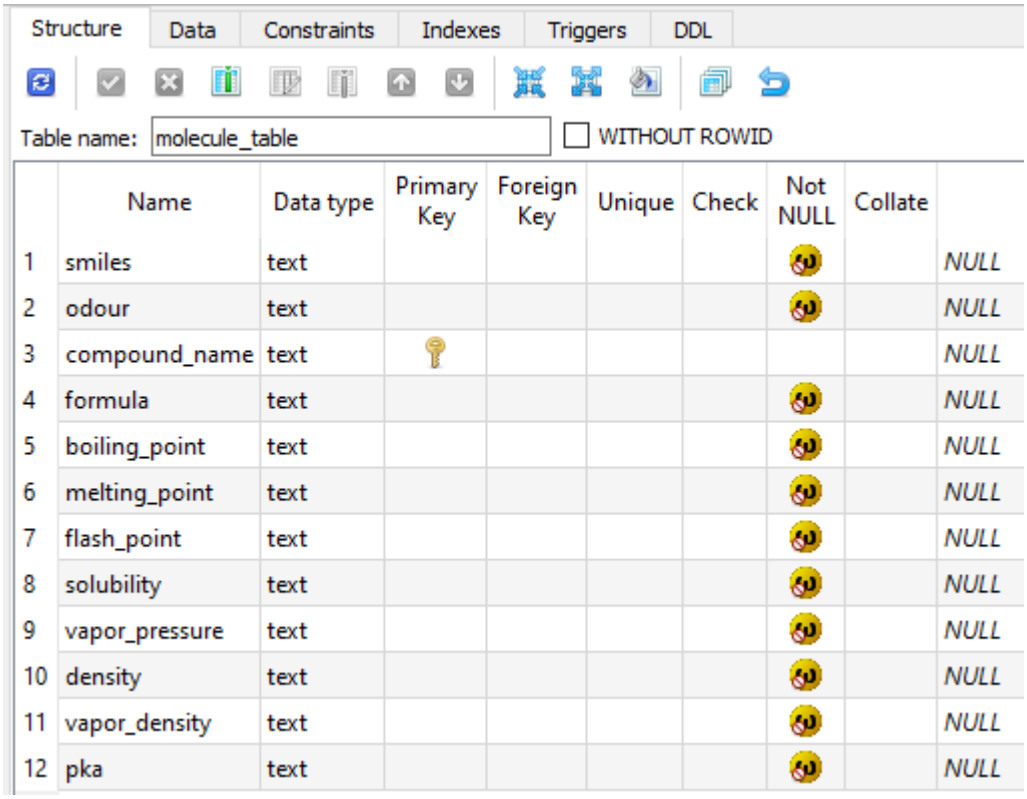
5.1 Banco de Dados

Combinando as informações de aromas (KORICHI, 2008) com os resultados obtidos na importação de dados do *PubChem*, pôde-se organizar o banco de dados de informações sobre os compostos com aroma, uma das finalidades deste trabalho.

Muito embora a fonte consultada forneça dados sobre odores para um número relativamente pequeno de compostos, o banco de dados elaborado conta com informações relevantes sobre os mesmos e pode contribuir para a obtenção de *insights* para determinados ramos, mesmo que limitados, da indústria. Além disso, como há uma funcionalidade no programa criado para a adição de novas entradas no banco, sua extensão não foi julgada como um problema por hora.

Utilizando o software *SQLiteStudio*, pode-se visualizar a estrutura do banco de dados:

Figura 2 - Estrutura do Banco de Dados de Compostos com Aroma



	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	
1	smiles	text							NULL
2	odour	text							NULL
3	compound_name	text							NULL
4	formula	text							NULL
5	boiling_point	text							NULL
6	melting_point	text							NULL
7	flash_point	text							NULL
8	solubility	text							NULL
9	vapor_pressure	text							NULL
10	density	text							NULL
11	vapor_density	text							NULL
12	pka	text							NULL

Fonte: Camargo e Eggers (2020).

Desta forma, estabelece-se que os valores únicos do banco de dados serão os nomes dos compostos – evidenciado pela atribuição do *status* de *primary key* à coluna relativa a eles. O grupo inicialmente idealizou utilizar a notação SMILES do composto para tal mas, considerando que o usuário teria mais chances de possuir o nome do composto que deseja adicionar do que seu string SMILES, optou-se pelo formato mostrado.

De forma a exibir a organização dos dados bem como exemplificar entradas no banco, o software *SQLiteStudio* também permite a visualização deste como tabela:

Figura 3 - Exemplo do Banco de Dados (Tabela Parcial)

Structure | Data | Constraints | Indexes | Triggers | DDL

Grid view | Form view

Filter data | Total rows loaded: 234

	smiles	odour	compound name	formula
1	O=C(OC2C(C)(C)C1CCC2(C)(C1))C	Doce	(1,3,3-trimethyl-2-bicyclo[2.2.1]heptanyl) acetate	C12H20O2
2	O=C(OC1CC2CCC1(C)C2(C)(C))C	Doce	(1,7,7-trimethyl-2-bicyclo[2.2.1]heptanyl) acetate	C12H20O2
3	NCc1ccc(cc1C)C	Doce	(2,4-dimethylphenyl)methanamine	C9H13N
4	O=C(OC1C(=CCC(C(=C)C)C1)C)CC	Doce	(2-methyl-5-prop-1-en-2-ylcyclohex-2-en-1-yl) propanoate	C13H20O2
5	O=Cc1ccc(OC(=O)C)c(OC)c1	Balsamo	(4-formyl-2-methoxyphenyl) acetate	C10H10O4
6	OCc1ccc(OC)cc1	Baunilha	(4-methoxyphenyl)methanol	C8H10O2
7	O=C(OCc1ccc(OC)cc1)C	Baunilha	(4-methoxyphenyl)methyl acetate	C10H12O3
8	O=C(OCc1ccc(OC)cc1)CC	Baunilha	(4-methoxyphenyl)methyl propanoate	C11H14O3
9	O=C(OC1CC(C)CCC1C(C)C)C	Doce	(5-methyl-2-propan-2-ylcyclohexyl) acetate	C12H22O2
10	O=C(c1ccc(cc1C)C)C	Doce	1-(2,4-dimethylphenyl)ethanone	C10H12O
11	O=C(c1ccc(OC)c(OC)c1)C	Doce	1-(3,4-dimethoxyphenyl)ethanone	C10H12O3
12	O=C(c1ccc(O)c(OC)c1)C	Baunilha	1-(4-hydroxy-3-methoxyphenyl)ethanone	C9H10O3
13	O=C(C)Cc1ccc(OC)cc1	Baunilha	1-(4-methoxyphenyl)propan-2-one	C10H12O2
14	O(c1ccccc1(OC))C	Baunilha	1,2-dimethoxybenzene	C8H10O2
15	O1C(C)(C)C2CCC1(C)CC2	Canfora	1,3,3-trimethyl-2-oxabicyclo[2.2.2]octane	C10H18O
16	O=Cc1ccc2OCOc2(c1)	Balsamo	1,3-benzodioxole-5-carbaldehyde	C8H6O3
17	OC1CC2CCC1(C)C2(C)(C)	Canfora	1,7,7-trimethylbicyclo[2.2.1]heptan-2-ol	C10H18O
18	O=C1CC2CCC1(C)C2(C)(C)	Canfora	1,7,7-trimethylbicyclo[2.2.1]heptan-2-one	C10H16O
19	O(c1ccc(cc1)C)C	Canfora	1-methoxy-4-methylbenzene	C8H10O
20	O(c1ccc(cc1)C=CC)C	Anis	1-methoxy-4-prop-1-enylbenzene	C10H12O
21	O(c1ccc(cc1)CC=C)C	Doce	1-methoxy-4-prop-2-enylbenzene	C10H12O
22	O(c1ccc(cc1)CCC)C	Anis	1-methoxy-4-propylbenzene	C10H14O
23	O1C2(C)(CCC1(CC2)C(C)C)	Canfora	1-methyl-4-propan-2-yl-7-oxabicyclo[2.2.1]heptane	C10H18O
24	O=C(c1ccccc1)CCC	Canfora	1-phenylbutan-1-one	C10H12O
25	O=C(c1ccccc1)C	Doce	1-phenylethanone	C8H8O
26	OCC(C)(C)CC(C)C	Frutado	2,2,4-trimethylpentan-1-ol	C8H18O
27	OC(C(C)C)C(C)(C)C	Canfora	2,2,4-trimethylpentan-3-ol	C8H18O
28	C=C2C1CCC(C1)C2(C)C	Canfora	2,2-dimethyl-3-methylidenebicyclo[2.2.1]heptane	C10H16
29	OCC(C)(C)CC	Canfora	2,2-dimethylbutan-1-ol	C6H14O
30	OC(CCC)C(C)(C)C	Canfora	2,2-dimethylhexan-3-ol	C8H18O
31	OCC(C)(C)CCC	Frutado	2,2-dimethylpentan-1-ol	C7H16O
32	OC(CC)C(C)(C)C	Canfora	2,2-dimethylpentan-3-ol	C7H16O

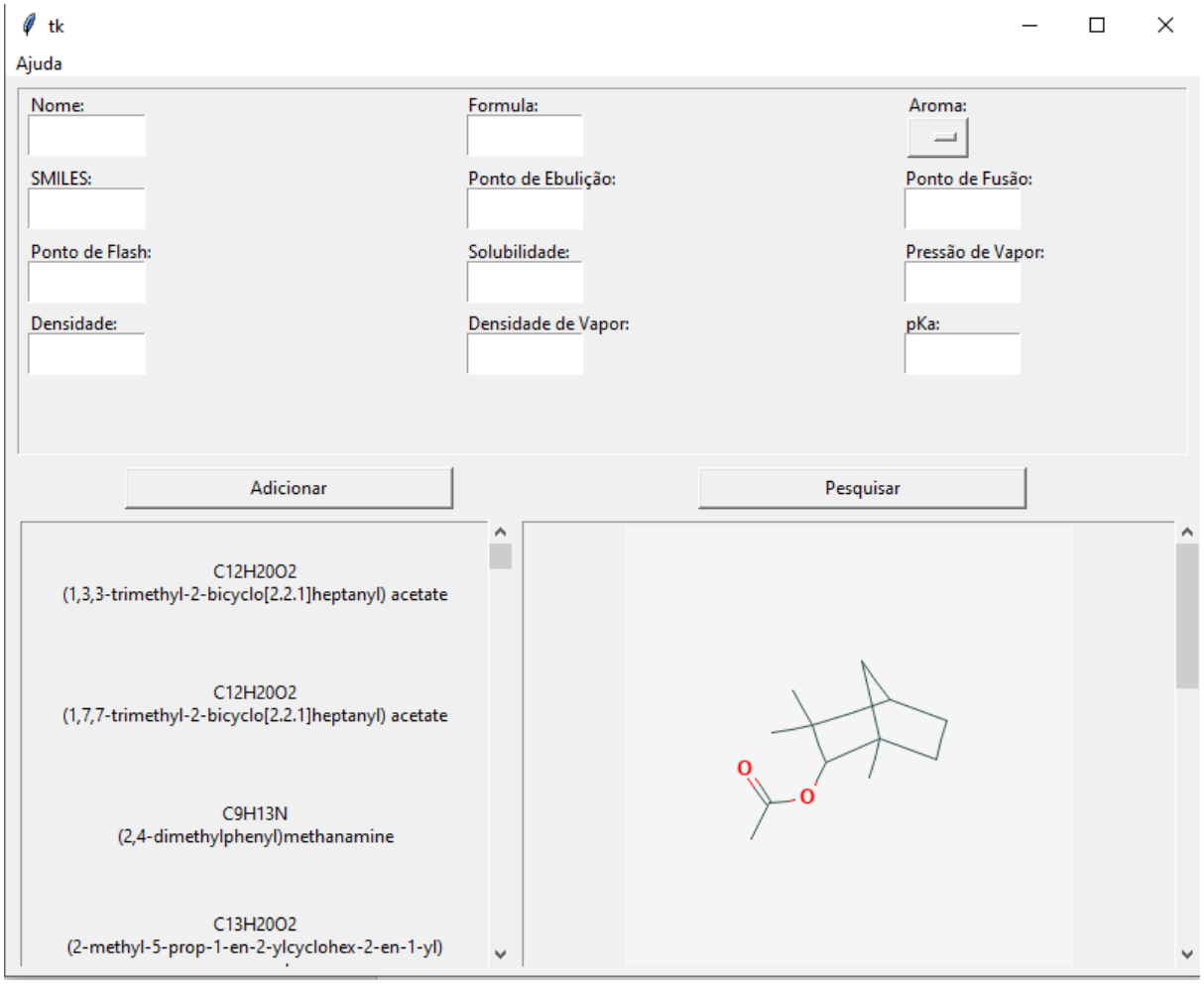
Fonte: Camargo e Eggers (2020).

5.2 Interface Gráfica

Através do programa em *Python* descrito na seção 4 desta monografia e exposto no apêndice, criou-se uma interface de usuário leve, simples e amigável. Julgou-se que as divisões da janela estão claras no sentido de separar as funcionalidades do programa. Desta forma, o usuário irá inserir compostos de interesse na seção superior da janela, selecionar o composto desejado na seção inferior esquerda e obter suas informações na seção inferior direita.

Ao se executar o programa, a visualização inicial do usuário será a seguinte:

Figura 4 - Visualização Inicial da Interface Gráfica

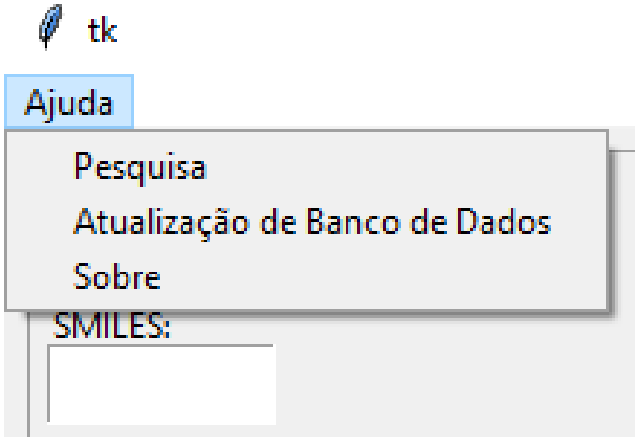


Fonte: Camargo e Eggers (2020).

Vale ressaltar que o programa é iniciado com resultados de uma consulta total do banco de dados. Desta maneira, o usuário já pode saber o formato em que os dados estão armazenados e fazer suas próprias consultas rapidamente.

Ainda assim, para evidenciar mais a funcionalidade da *GUI*, o usuário pode acessar o menu “Ajuda” no topo da página, os quais contam com exemplos de execução:

Figura 5 - Menu de Ajuda ao Usuário



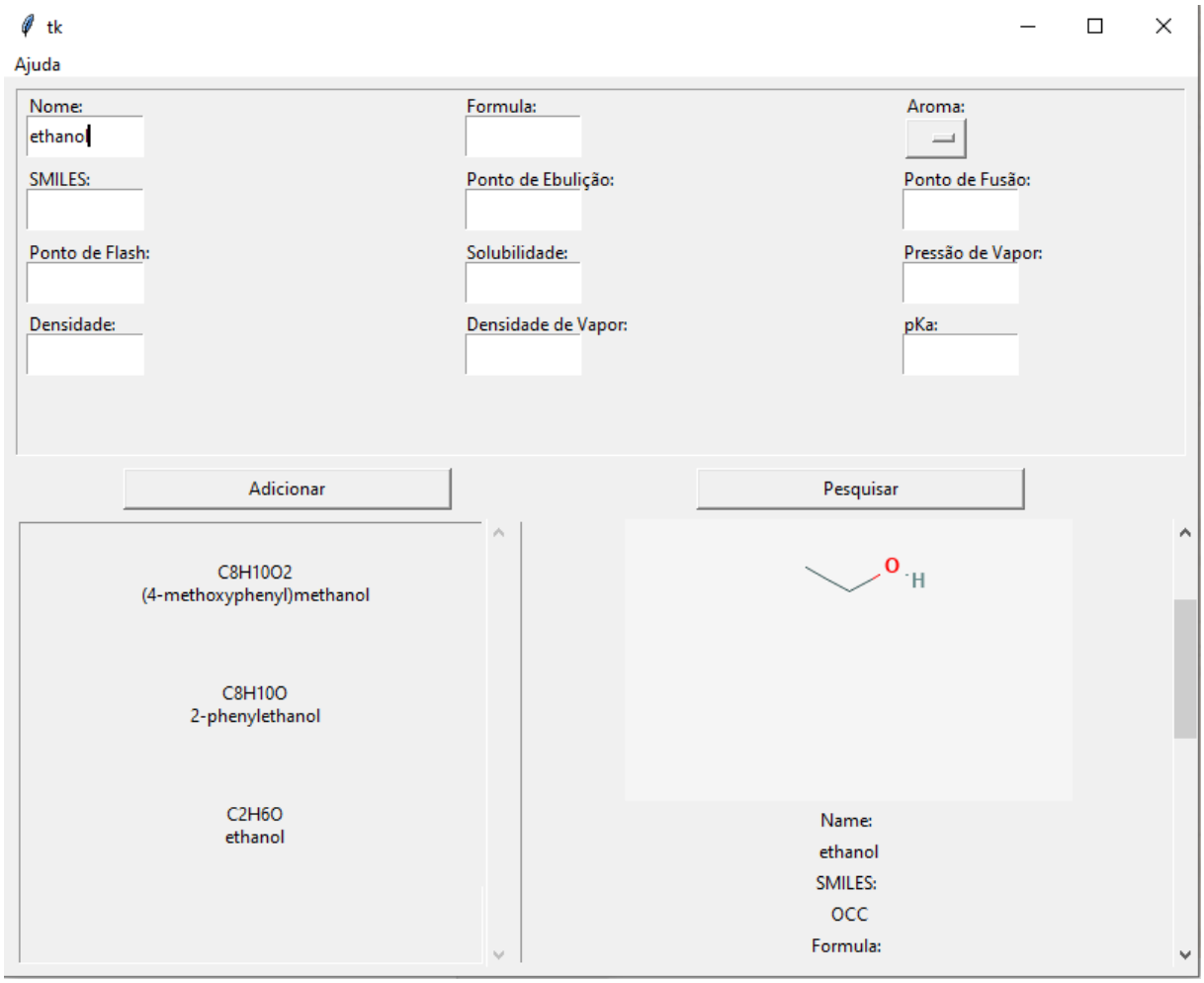
Fonte: Camargo e Eggers (2020).

5.2.1 Pesquisa de Compostos

Para realizar pesquisas ao banco de dados existente, o programa usa uma *query* com o operador LIKE. Um exemplo de pesquisa neste formato é “*SELECT * FROM molecule_table WHERE compound_name LIKE ‘%ethanol%’*”. Esta pesquisa retornaria todas as correspondências no banco de dados cujo valor da coluna “*compound_name”* contenha “*ethanol"*, incluindo correspondências parciais.

Desta maneira, o programa torna possível pesquisas mais avançadas no qual a entrada do usuário segue um padrão mas não representa o valor completo no banco de dados. Por exemplo, ao executar a consulta exposta acima, o programa retorna na *GUI* os seguintes resultados:

Figura 6 - Resultado da pesquisa com nome de composto contendo "ethanol"



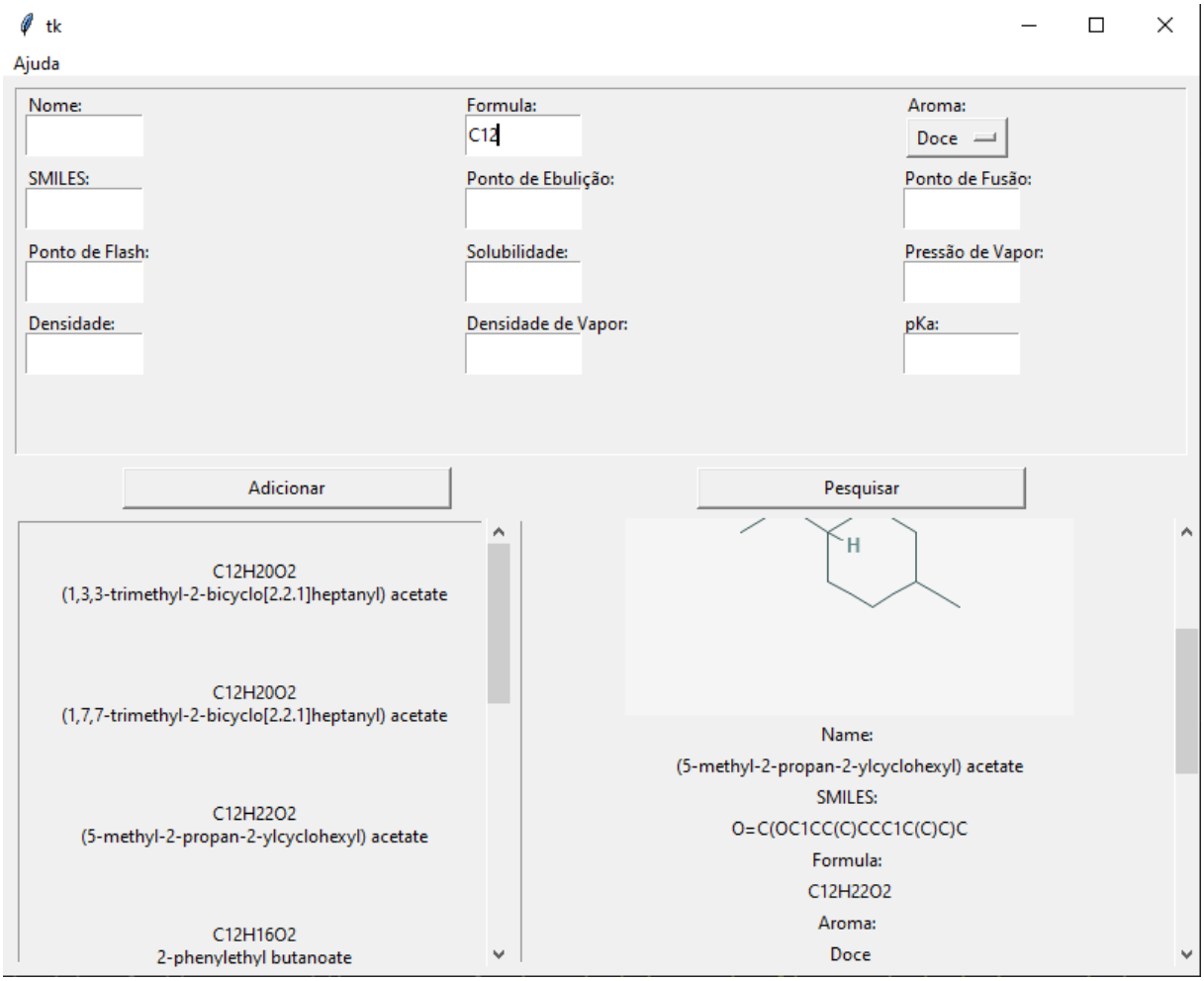
Fonte: Camargo e Eggers (2020).

Conforme mostrado na figura 6, a consulta retorna todas as correspondências parciais a “ethanol” no banco de dados.

Uma funcionalidade interessante advinda do uso do operador LIKE é a possibilidade de se pesquisar compostos com um determinado número de átomos de carbonos (ou outro elemento) em sua estrutura no campo de “Fórmula”.

Por exemplo, caso o usuário queira pesquisar aromas de compostos com 12 átomos de carbono com aroma doce, pode realizar a seguinte pesquisa:

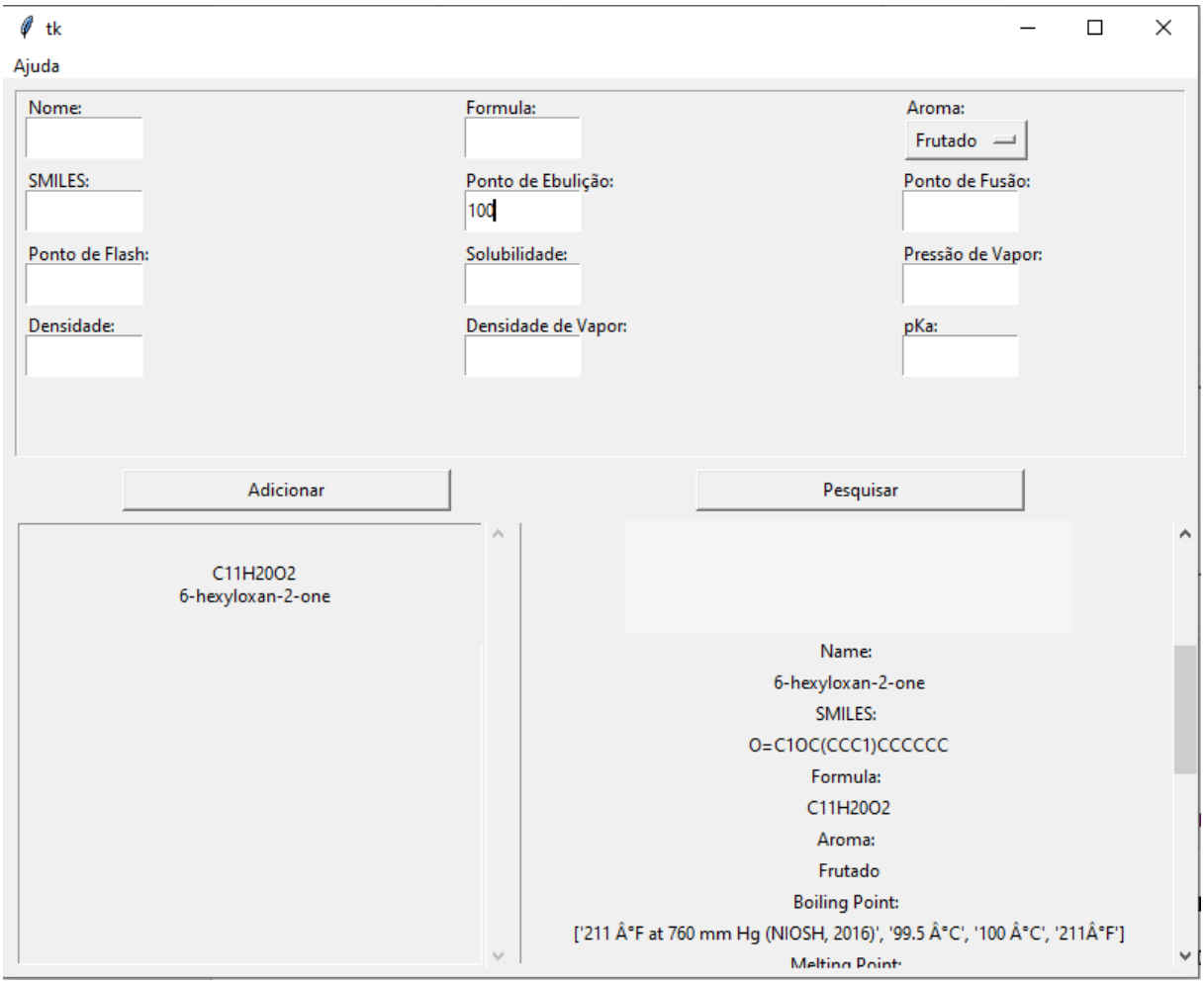
Figura 7 - Resultado da pesquisa de compostos com 12 carbonos e aroma doce



Fonte: Camargo e Eggers (2020).

Ademais, é possível pesquisar o banco de dados utilizando qualquer uma das entradas de propriedades. Por exemplo, para pesquisar compostos cuja entrada no banco de dados relativa ao ponto de ebulição contenha “100” e tenha aroma frutado, basta realizar a seguinte pesquisa:

Figura 8 - Resultado da pesquisa de compostos com ponto de ebulição contendo "100" e aroma frutado



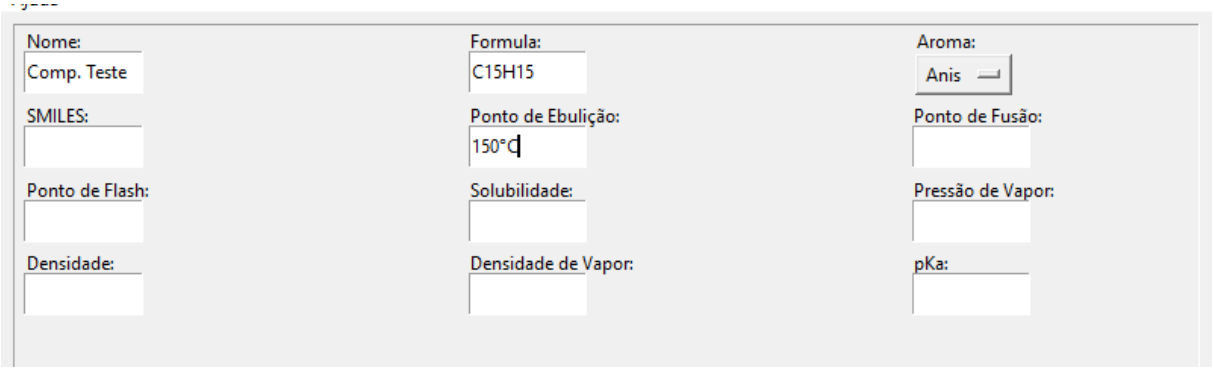
Fonte: Camargo e Eggers (2020).

5.2.2 Adição de Compostos

Para que o usuário possa popular o banco de dados com informações de compostos próprios, é necessário que ele preencha o campo de nome (obrigatório, uma vez que este é a *primary key* do banco de dados) e pelo menos um outro campo de propriedades. Caso a entrada do usuário não seja neste formato, o composto não é adicionado.

Um exemplo de entrada para a adição de compostos pode ser o seguinte:

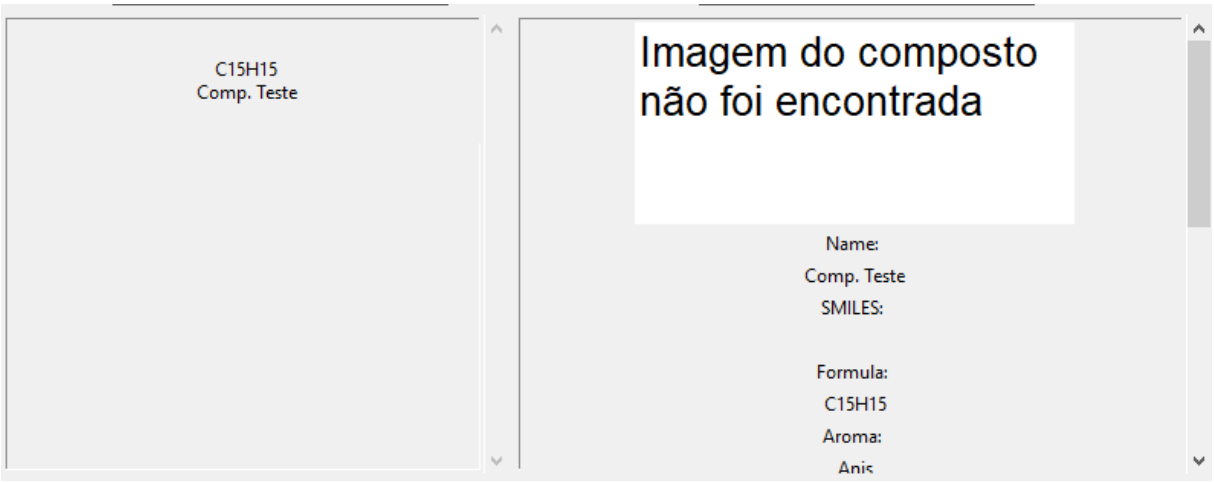
Figura 9 - Exemplo de uma entrada válida para adição de composto



Fonte: Camargo e Eggers (2020).

Após clicar no botão “Adicionar”, o composto estará no banco de dados e poderá ser buscado normalmente. O resultado da pesquisa confirma que o composto de fato foi incluído:

Figura 10 - Resultado da pesquisa do composto adicionado ao banco de dados



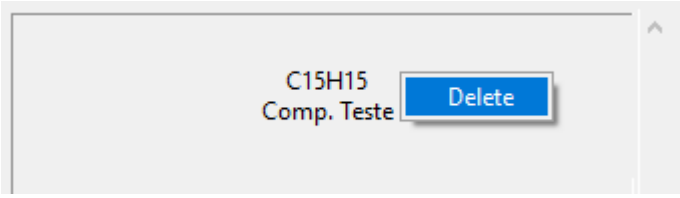
Fonte: Camargo e Eggers (2020).

Note que o composto não tem imagem, uma vez que estas foram buscadas no *PubChem*. Além disso, é necessário ter a representação SMILES para inserir a imagem da estrutura do composto na saída do programa, pois estas são nomeadas com este valor na pasta “images”.

5.2.3 Remoção de Compostos

Além da adição de compostos ao banco de dados, o usuário pode também removê-los de maneira simples. Para remover o composto adicionado no item 5.2.1 desta monografia, basta clicar com o botão direito no *hyperlink* do composto em questão e clicar em “Delete”, tal como evidenciado na figura a seguir:

Figura 11 - Exemplo da remoção de um composto



Fonte: Camargo e Eggers (2020).

6 TRABALHOS FUTUROS

Durante o desenvolvimento deste trabalho, o grupo idealizou diversas funcionalidades que agregariam valor ao programa final. Nesta seção, estão apresentadas estas ideias de forma que eventuais desenvolvimentos futuros possam inclui-las.

6.1 Ampliação do Banco de Dados

O banco de dados atual, embora seja uma base com propriedades válidas, ainda é relativamente pequeno – tanto no sentido de número de compostos quanto de propriedades – para aplicações comerciais. Há apenas 234 compostos no banco, tornando-o consideravelmente restrito.

A adição de novos compostos está coberta no funcionamento do programa com o método *add_compounds()* da classe *Window* da interface gráfica, mas é restrita a um composto por vez. Sua modificação ou até mesmo a criação de um novo método de atualização por lotes de dados – funcionalidade já suportada pelo *SQLite* na versão 3.7.1.1 (SQLITE CONSORTIUM) – poderia aprimorar o programa. Este método poderia receber, por exemplo, um arquivo no formato correto do banco de dados com os compostos a serem adicionados para acelerar a atualização da base.

A adição de novas propriedades requer a criação de novas colunas na tabela de compostos presente no banco de dados. Embora não abordada neste trabalho, tal funcionalidade poderia ser adicionada através do comando “ALTER TABLE ... ADD ...” do SQLite.

Entretanto, a adição de novas colunas necessitaria a inclusão de mais barras de pesquisa na interface gráfica, necessitando uma alteração no seu *layout*. Por isso, também seria preciso incluir no código *app.py* alguma propriedade responsável por adicionar dinamicamente *widgets* relacionados às colunas, alterando a distribuição de objetos na janela automaticamente.

6.2 Exportação de dados consultados

Tal como presente na interface do banco de dados para óleos vegetais desenvolvido pelo orientador desse trabalho, a *GUI* para o banco de dados de compostos com aroma apresentada neste trabalho poderia conter uma funcionalidade para exportar o resultado das consultas ao banco de dados pelo usuário.

Formatos relevantes para o uso das informações retornadas pela pesquisa seriam:

- *.txt*, facilmente criado com funções e métodos nativos ao *Python* – a função *open()* para abrir um arquivo, o método *write()* para adicionar texto ao arquivo e o método *close()* para fechá-lo;

- Formatos do *Excel (.xls* e *.xlsx)*, uma vez que este é o software de manipulação de dados mais usado no mundo. A escrita de arquivos nestes formatos é abordada no arquivo *import_pubchem* apresentado neste trabalho. O processo envolveria o uso do módulo *xlsxwriter*, havendo a criação de um objeto da classe *xlsxwriter.Workbook*, a adição de uma planilha com o método *add_worksheet()*, a escrita dos dados com o método *write()* e o subsequente fechamento da pasta de trabalho com o método *.close()*

- *.csv*, pois este é um formato com tamanho relativamente pequeno, com fácil leitura e escrita. A criação de um arquivo neste formato envolveria o uso do módulo *csv* do *Python*, havendo a criação de um objeto *csv.writer* e a escrita de dados com a método *writerow().*

6.3 Adição de Tabelas de Coeficientes

Na forma como o programa está estruturado, as propriedades retornadas pelas consultas são relativas à condições – como temperaturas e pressões – fixas. Em processos industriais, no entanto, é necessário ter mais flexibilidade nestes valores, calculando as propriedades nas condições variáveis dos processos.

Para isso, o banco de dados poderia conter uma nova tabela relativa a coeficientes de modelos de propriedades. Por exemplo, para a capacidade térmica, entalpia e entropia, tem-se as seguintes equações empíricas genéricas (NASA, 1993):

$$Capacidade\ de\ Calor\colon \frac{C_p^o(T)}{R} = a_1 + a_2T + a_3T^2 + a_4T^3 + a_5T^4$$

$$Entalpia\colon \frac{H^o(T)}{RT} = a_1 + a_2\frac{T}{2} + a_3\frac{T^2}{3} + a_4\frac{T^3}{4} + a_5\frac{T^4}{5} + \frac{b_1}{T}$$

$$Entropia\colon \frac{S^o(T)}{R} = a_1 \ln T + a_2T + a_3\frac{T^2}{2} + a_4\frac{T^3}{3} + a_5\frac{T^4}{4} + b_2$$

Tendo uma tabela com os coeficientes $a_i\ e\ b_i$ tornaria possível o cálculo destas propriedades para um composto em uma determinada temperatura.

7 CONCLUSÕES

O programa criado, embora atualmente não forneça *insights* completa e imediatamente aplicáveis à indústria cosmética, pode servir como uma base consistente para tal. Com os valores iniciais de propriedades no banco de dados e a possibilidade de o usuário adicionar mais compostos, tomou-se um primeiro passo importante para a coleta de informações essencial para uma eventual extensão do programa à uma aplicação de CAMD.

Além disso, notou-se que a criação e modificação do banco de dados, bem como as pesquisas a ele, são feitas de maneira rápida e objetiva com a linguagem *SQLite* e sua interação com o *Python*. A interface de usuário, embora simples, mostrou-se suficiente para atender as demandas iniciais por consultas a um banco de dados. Pôde-se concluir, com isto, que é possível, viável, rápido e eficiente integrar estas linguagens para criar aplicações com interações entre GUI e bancos de dados.

Para atingir funcionalidades mais amplas e dinâmicas, recomenda-se incluir os tópicos apresentados no capítulo 5 deste trabalho futuramente. Desta forma, a ferramenta criada poderá fornecer mais informações para profissionais da indústria, sobretudo a de cosméticos, e auxiliar no desenvolvimento de novas fragrâncias e produtos.

8 REFERÊNCIAS BIBLIOGRÁFICAS

OSTERATH, Brigitte. Artificial intelligence creates perfumes without being able to smell them. **Deutsche Welle**, 31 mai. 2019. Disponível em: https://www.dw.com/en/artificial-intelligence-creates-perfumes-without-being-able-to-smell-them/a-48989202. Acesso em: 8 out. 2019.

IN-COSMETICS HAMBURG, 2014, Hamburgo. **Fragrance in Cosmetics – The Smell of Success** [...]. *[S. l.: s. n.]*, 2014. Disponível em: https://bit.ly/2Xn4A0s. Acesso em: 9 abr. 2020.

CLARK, Paul. Running Head: Management Overview of Scent as a Marketing Communications Tool. Suíça, 2009.

Euro Cosmetics. In: OETTERER, Enilce. Exploring the Brazilian Fragrance Market: A View of Consumer Habits, Preferences and Technology. *[S. l.]*. 2015. Disponível em: http://www.abc-cosmetologia.org.br/wp-content/uploads/2015/08/ENILCE_EC0115.pdf

UNIFIED MODELING LANGUAGE. *[S. l.], [S. d.]*. Disponível em https://www.uml.org/index.htm. Acesso em 9 abr. 2020.

KORICHI, Mourad. et. al. Computer-aided aroma design. II. Quantitative structure–odour relationship. **Chemical Engineering and Processing: Process Intensification**, volume 47, ed. 11, 1912-1925, outubro de 2008.

SWAIN, Matt. **PubChemPy Documentation**: Release 1.0.4. [*S. l.*], 11 abr. 2017. Disponível em: https://pubchempy.readthedocs.io/en/latest/. Acesso em: 2 dez. 2019

PYTHON. **tkinter - Python interface to Tcl/Tk - Python 3.8.2 documentation**, 2020. Disponível em: <https://docs.python.org/3/library/tkinter.html>. Acesso em: 17 de abr. de 2020.

CLARK, Alex. **Pillow - Pillow (PIL Fork) 7.1.1 documentation**, 2020. Disponível em: <https://pillow.readthedocs.io/en/stable/>. Acesso em: 17 de abr. de 2020.

PYTHON. **sqlite3- DB-API 2.0 interface for SQLite databases**, 2020. Disponível em: <https://docs.python.org/3/library/sqlite3.html>. Acesso em: 17 de abr. de 2020.

PYTHON. **csv — CSV File Reading and Writing**, 2020. Disponível em: <https://docs.python.org/3/library/csv.html>. Acesso em: 17 de abr. de 2020.

PYTHON. **time — Time access and conversions**, 2020. Disponível em: <https://docs.python.org/3/library/time.html>. Acesso em: 17 de abr. de 2020.

PYTHON. **os — Miscellaneous operating system interfaces**, 2020. Disponível em: <https://docs.python.org/3/library/os.html>. Acesso em: 17 de abr. de 2020.

MUTHUKADAN, Baiju. **Selenium with Python, 2020**. Documentação da biblioteca selenium. Disponível em: <https://selenium-python.readthedocs.io/>. Acesso em: 17 de abr. de 2020.

REITZ, Kenneth. Requests: HTTP for Humans, 2020. Disponível em: <https://requests.readthedocs.io/en/master/>. Acesso em: 17 de abr. de 2020.

MCNAMARA, John. **Creating Excel files with Python and XlsxWriter**, 2020. Disponível em: <https://xlsxwriter.readthedocs.io/index.html>. Acesso em: 17 de abr. de 2020

SQLITE CONSORTIUM. **What Is SQLite?**. [*S. l.*]. Disponível em: https://www.sqlite.org/index.html. Acesso em: 7 dez. 2019.

SQLITE CONSORTIUM. **Implementation limits for SQLite.** *[S. l].* Disponível em: https://www.sqlite.org/limits.html. Acesso em 9 abr. 2020

SQLITE CONSORTIUM. **SQL as Understood By SQLite – Insert.** [*S. l.*]. Disponível em: https://www.sqlite.org/lang_insert.html. Acesso em: 11 mar. 2020.

BHAWANI. The Incredible Growth of Python Language. **Zibtek**. [*S. l.*], 22 out. 2019. Disponível em: https://www.zibtek.com/blog/the-incredible-growth-of-python/. Acesso em: 7 dez. 2019.

KUMAR, Yogesh. et. al. AromaDb: A Database of Medicinal and Aromatic Plant's Aroma Molecules With Phytochemistry and Therapeutic Potentials. **Frontiers in Plant Science.** [*S. l.*], 13 ago. 2018. Disponível em: https://www.frontiersin.org/articles/10.3389/fpls.2018.01081/full. Acesso em: 21 jan. 2019.

ACREE, Terry; ARN, Heinrich. **FlavorNet**. [*S. l.*], 2004. Disponível em: http://flavornet.org/flavornet.html. Acesso em: 11 fev. 2020.

GOOD SCENTS COMPANY (Oak Creek, WI). **The Good Scents Company Information System**. [*S. l.*], 1994. Disponível em: http://www.thegoodscentscompany.com/. Acesso em: 11 fev. 2020.

TELES DOS SANTOS, M.; OLIVEIRA, C.C.; LE ROUX, G. A. C. Desenvolvimento de Banco de Dados de Óleos Vegetais para Aplicação em Projeto de Produtos. São Paulo, 2014.

VALLI, M.; et. al. Development of a Natural Products Database from the Biodiversity of Brazil. **Journal of Natural Products**. EUA, 2013, Ed. 76, pg. 439-444, 18 jan. 2013.

NASA - National Aeronautics and Space Administration. In: J. MCBRIDE, Bonnie; GORDON, Sanford; A. RENO, Martin. Coefficients for Calculating Thermodynamic and Transport Properties of Individual Species. [S. l.: s. n.], 1993.

9 APÊNDICE

9.1 Programa em Python para a GUI *(app.py)*

```
import tkinter as tk
import query
from PIL import Image, ImageTk
import os
from tkinter import messagebox


CURR_PATH = os.path.dirname(__file__)

ODOURS = [
    "Anis",
    "Azedo",
    "Balnilha",
    "Balsamo",
    "Canfora",
    "Doce",
    "Frutado",
    "Mel",
    "Rosa",
]

ODOURS = query.get_odours()


class App(tk.Frame):
    def __init__(self, parent, *args, **kwargs):
        tk.Frame.__init__(self, parent, *args, **kwargs)

        self.parent = parent
        self.results_buttons = []

        self.create_search_frame(self.parent)

        self.create_results_frame(self.parent)

        self.create_image_frame(self.parent)

        self.create_popup_menu(self.parent)

    def search(self):
        self.submit()
        self.update_results_frame()

    def add_compound(self):
        result = tk.messagebox.askokcancel(
            message="Do you want to add the compound's information?"
        )
        if result:
            add_compound_info = [
                (
                    self.smiles_search_bar.get(),
                    self.odour_dropbox_value.get(),
                    self.compound_name_search_bar.get(),
                    self.formula_search_bar.get(),
                    self.boiling_point_search_bar.get(),
                    self.melting_point_search_bar.get(),
                    self.flash_point_search_bar.get(),
                    self.solubility_search_bar.get(),
                    self.vapor_pressure_search_bar.get(),
                    self.density_search_bar.get(),
                    self.vapor_density_search_bar.get(),
```

```
                    self.pka_search_bar.get(),
                )
            ]
            # button goes down
            self.add_button.configure(relief="sunken")

            # checa se o usuário colocou nome e alguma outra propriedade
            if add_compound_info[0][2] == "" or True not in [
                x != ""
                for x in add_compound_info[0]
                if add_compound_info[0].index(x) != 2
            ]:
                messagebox.showerror(
                    message="O composto deve ter um nome e pelo menos outra propriedade. Por f
avor, verifique."
                )
            else:
                query.update_db(add_compound_info)

                # clears the entry search bars
                self.compound_name_search_bar.delete(0, "end")
                self.smiles_search_bar.delete(0, "end")
                self.formula_search_bar.delete(0, "end")
                self.boiling_point_search_bar.delete(0, "end")
                self.melting_point_search_bar.delete(0, "end")
                self.flash_point_search_bar.delete(0, "end")
                self.solubility_search_bar.delete(0, "end")
                self.vapor_pressure_search_bar.delete(0, "end")
                self.density_search_bar.delete(0, "end")
                self.vapor_density_search_bar.delete(0, "end")
                self.pka_search_bar.delete(0, "end")
                self.odour_dropbox_value.set(ODOURS[0])

                messagebox.showinfo(
                    "Atualização de dados", "Composto adicionado com sucesso."
                )

            # button comes up again
            self.add_button.configure(relief="raised")
            return "break"

    def create_search_frame(self, parent):

        self.search_frame = tk.Frame(parent, bd=1, relief=tk.SUNKEN)
        self.button_frame = tk.Frame(parent)

        self.compound_name_search_bar = self._create_search_bar(
            self.search_frame, x=0.00625, y=0, label="Nome: "
        )
        self.formula_search_bar = self._create_search_bar(
            self.search_frame, x=0.38125, y=0, label="Formula: "
        )
        tk.Label(self.search_frame, text="Aroma: ").place(relx=0.75875, rely=0)
        self.odour_dropbox, self.odour_dropbox_value = self.create_dropbox(
            self.search_frame, ODOURS
        )
        self.odour_dropbox.place(relx=0.75875, rely=0.07)

        self.smiles_search_bar = self._create_search_bar(
            self.search_frame, x=0.00625, y=0.2, label="SMILES: "
        )
        self.boiling_point_search_bar = self._create_search_bar(
            self.search_frame, x=0.38125, y=0.2, label="Ponto de Ebulição: "
        )
        self.melting_point_search_bar = self._create_search_bar(
            self.search_frame, x=0.75625, y=0.2, label="Ponto de Fusão: "
        )
        self.flash_point_search_bar = self._create_search_bar(
            self.search_frame, x=0.00625, y=0.4, label="Ponto de Flash: "
        )
        self.solubility_search_bar = self._create_search_bar(
```

```
            self.search_frame, x=0.38125, y=0.4, label="Solubilidade: "
        )
        self.vapor_pressure_search_bar = self._create_search_bar(
            self.search_frame, x=0.75625, y=0.4, label="Pressão de Vapor: "
        )
        self.density_search_bar = self._create_search_bar(
            self.search_frame, x=0.00625, y=0.6, label="Densidade: "
        )
        self.vapor_density_search_bar = self._create_search_bar(
            self.search_frame, x=0.38125, y=0.6, label="Densidade de Vapor: "
        )
        self.pka_search_bar = self._create_search_bar(
            self.search_frame, x=0.75625, y=0.6, label="pKa: "
        )

        self.submit_button = tk.Button(
            self.button_frame, text="Pesquisar", command=self.search
        )
        self.submit_button.place(relx=0.58, rely=0.2, height=28, relwidth=0.275)

        self.add_button = tk.Button(
            master=self.button_frame, text="Adicionar", command=self.add_compound
        )
        self.add_button.place(relx=0.1, rely=0.2, height=28, relwidth=0.275)

        self.search_frame.place(relx=0.01, rely=0.01, relwidth=0.98, relheight=0.41)
        self.button_frame.place(relx=0, rely=0.42, relwidth=1, relheight=0.07)

        parent.bind(
            "<Return>", self.submit
        )  # hitting the enter button has the same function as clicking the 'Pesquisar' button

    def _create_search_bar(self, parent, x, y, label, relwidth=0.1, height=28):
        label = tk.Label(parent, text=label)
        label.place(relx=x, rely=y)

        search_bar = tk.Entry(parent, fg="black")
        search_bar.place(
            relx=x + 0.002, rely=y + 0.07, relwidth=relwidth, height=height
        )

        return search_bar

    def create_dropbox(self, parent, options):
        value = tk.StringVar(parent)
        value.set(options[0])

        return tk.OptionMenu(parent, value, *options), value

    def create_results_frame(self, parent):
        self.results_frame = ScrollFrame(parent)
        self.results_frame.bind("<Enter>", self._frame_entered)
        self.results_frame.bind("<Leave>", self._frame_left)
        self.submit()
        self.update_results_frame()

        self.results_frame.place(relx=0.01, rely=0.49, relheight=0.5, relwidth=0.415)

    def create_image_frame(self, parent):
        self.image_frame = ScrollFrame(parent)
        self.image_frame.bind("<Enter>", self._frame_entered)
        self.image_frame.bind("<Leave>", self._frame_left)
        self.image_panel = tk.Label(master=self.image_frame.viewPort)
        self.image_panel.pack()

        tk.Label(self.image_frame.viewPort, text="Name: ").pack()
        self.result_name = WrappingLabel(self.image_frame.viewPort, text="")
        self.result_name.pack()

        tk.Label(self.image_frame.viewPort, text="SMILES: ").pack()
        self.result_smiles = WrappingLabel(self.image_frame.viewPort, text="")
```

```
        self.result_smiles.pack()

        tk.Label(self.image_frame.viewPort, text="Formula: ").pack()
        self.result_formula = WrappingLabel(self.image_frame.viewPort, text="")
        self.result_formula.pack()

        tk.Label(self.image_frame.viewPort, text="Aroma: ").pack()
        self.result_odour = WrappingLabel(self.image_frame.viewPort, text="")
        self.result_odour.pack()

        tk.Label(self.image_frame.viewPort, text="Boiling Point: ").pack()
        self.result_boiling_point = WrappingLabel(self.image_frame.viewPort, text="")
        self.result_boiling_point.pack()

        tk.Label(self.image_frame.viewPort, text="Melting Point: ").pack()
        self.result_melting_point = WrappingLabel(self.image_frame.viewPort, text="")
        self.result_melting_point.pack()

        tk.Label(self.image_frame.viewPort, text="Flash Point: ").pack()
        self.result_flash_point = WrappingLabel(self.image_frame.viewPort, text="")
        self.result_flash_point.pack()

        tk.Label(self.image_frame.viewPort, text="Solubility: ").pack()
        self.result_solubility = WrappingLabel(self.image_frame.viewPort, text="")
        self.result_solubility.pack()

        tk.Label(self.image_frame.viewPort, text="Vapor Pressure: ").pack()
        self.result_vapor_pressure = WrappingLabel(self.image_frame.viewPort, text="")
        self.result_vapor_pressure.pack()

        tk.Label(self.image_frame.viewPort, text="Density: ").pack()
        self.result_density = WrappingLabel(self.image_frame.viewPort, text="")
        self.result_density.pack()

        tk.Label(self.image_frame.viewPort, text="Vapor Density: ").pack()
        self.result_vapor_density = WrappingLabel(self.image_frame.viewPort, text="")
        self.result_vapor_density.pack()

        tk.Label(self.image_frame.viewPort, text="pKa: ").pack()
        self.result_pka = WrappingLabel(self.image_frame.viewPort, text="")
        self.result_pka.pack()

        self.update_image_frame(
            self.displayed_values[0]
        )  # when first creating the image frame, show the image of the first compound on the
list

        self.image_frame.place(relx=0.43, rely=0.49, relheight=0.5, relwidth=0.57)

    def create_popup_menu(self, parent):
        self.popup_menu = tk.Menu(parent, tearoff=0)
        self.popup_menu.add_command(
            label="Delete", command=self.delete_compound
        )  # adds the delete option to the menu which calls delete_compound by clicking it

    def update_image_frame(self, compound_info):

        if os.path.isfile(os.path.join(CURR_PATH, "images", compound_info[0]) + ".png"):

            img = tk.PhotoImage(
                file=f'{os.path.join(CURR_PATH, "images", compound_info[0])}.png'
            )

        else:

            img = tk.PhotoImage(
                file=f'{os.path.join(CURR_PATH, "images", "notfound")}.png'
            )

        self.image_panel.configure(image=img)
        self.image_panel.image = img
```

```
        self.result_smiles.configure(text=compound_info[0])
        self.result_odour.configure(text=compound_info[1])
        self.result_name.configure(text=compound_info[2])
        self.result_formula.configure(text=compound_info[3])
        self.result_boiling_point.configure(text=compound_info[4])
        self.result_melting_point.configure(text=compound_info[5])
        self.result_flash_point.configure(text=compound_info[6])
        self.result_solubility.configure(text=compound_info[7])
        self.result_vapor_pressure.configure(text=compound_info[8])
        self.result_density.configure(text=compound_info[9])
        self.result_vapor_density.configure(text=compound_info[10])
        self.result_pka.configure(text=compound_info[11])

    def submit(self, event=None):
        self.displayed_values = query.get_data(
            smiles=self.smiles_search_bar.get(),
            compound_name=self.compound_name_search_bar.get(),
            formula=self.formula_search_bar.get(),
            boiling_point=self.boiling_point_search_bar.get(),
            melting_point=self.melting_point_search_bar.get(),
            flash_point=self.flash_point_search_bar.get(),
            solubility=self.solubility_search_bar.get(),
            vapor_pressure=self.vapor_pressure_search_bar.get(),
            density=self.density_search_bar.get(),
            vapor_density=self.vapor_density_search_bar.get(),
            pka=self.pka_search_bar.get(),
            odour=self.odour_dropbox_value.get(),
        )

    def delete_compound(self):
        if self.deleted_compound:
            deleted = query.delete_compound(
                self.deleted_compound[2]
            )  # deletes compound from the database, returns True if deleted successfully
            if deleted:
                tk.messagebox.showinfo(
                    "Deletion!", f"Deleted {self.deleted_compound[2]} SUCCESSFULLY"
                )
                self.submit()
                self.update_results_frame()  # submit and update_results_frame are called to r
eset the results display
            else:
                tk.messagebox.showerror(
                    "Deletion!",
                    f"Something went wrong when trying to delete {self.deleted_compound[2]}. D
eletion FAILED",
                )
        else:
            tk.messagebox.showerror(
                "Deletion!",
                "Could not locate the compound on the database, try refreshing the compound li
st by clicking 'Pesquisar' again.",
            )  # in a case where deleted_compound was not set, the compound problably doesnt e
xist anymore

    def _handle_result_button_click(self, event):
        self.update_image_frame(event.widget.info)

    def update_results_frame(self):
        for compound_buttons in self.results_buttons:
            compound_buttons.grid_forget()  # removes the buttons before adding new ones
        self.results_buttons = (
            []
        )  # clears the list of buttons since the buttons have been deleted
        for row in self.displayed_values:
            result = WrappingLabel(
                self.results_frame.viewPort,
                text=f"{row[3]}\n{row[2]}",
                cursor="hand2",
                height=5,
                width=45,
```

```
            )  # creates a button for each result of the query

            result.info = row
            result.grid(row=self.displayed_values.index(row), sticky="nsew")
            self.results_frame.viewPort.grid_columnconfigure(
                0, weight=1
            )  # makes the labels fill the scrollable frame if the main window is resized
            result.bind(
                "<Button-1>", lambda x: self._handle_result_button_click(x),
            )
            result.bind(
                "<Button-3>", lambda x: self._handle_result_button_right_click(x)
            )  # binds the right button to the _handle_result_button_right_click
            self.results_buttons.append(
                result
            )  # adds them to the list so we know to clean them up later

    # once the mouse is inside the frame bound to this command, binds the canvas portion of th
e frame to a handle scroll function using the mouse wheel
    def _frame_entered(self, event):
        event.widget.canvas.bind_all(
            "<MouseWheel>", lambda x: self._handle_scroll(x, event.widget.canvas)
        )  # passes the canvas to the _handle_scroll function because the scroll event can hap
pen over any widget inside the frame entered, but only the canvas can actually yview_scroll

    # once the mouse leaves the frame bound to this command, using the mouse whell wont do any
thing
    def _frame_left(self, event):
        event.widget.canvas.unbind_all("<MouseWheel>")

    # scrolls a canvas yview
    def _handle_scroll(self, event, canvas):
        canvas.yview_scroll(int(-1 * (event.delta / 120)), "units")

    def _handle_result_button_right_click(self, event):
        self.deleted_compound = event.widget.info
        self.popup_menu.post(
            event.x_root, event.y_root
        )  # opens the popup menu with the delete option where the mouse clicked with the righ
t button

class WrappingLabel(tk.Label):

    def __init__(self, master, **kwargs):
        tk.Label.__init__(self, master, **kwargs)
        self.bind('<Configure>', lambda e: self.config(wraplength=master.winfo_width()))


class ScrollFrame(tk.Frame):
    def __init__(self, parent):
        super().__init__(parent)  # create a frame (self)

        self.canvas = tk.Canvas(
            self, borderwidth=0, background="#F0F0F0", width=150, bd=1, relief=tk.SUNKEN
        )  # place canvas on self
        self.viewPort = tk.Frame(
            self.canvas, background="#F0F0F0"
        )  # place a frame on the canvas, this frame will hold the child widgets
        self.vsb = tk.Scrollbar(
            self, orient="vertical", command=self.canvas.yview
        )  # place a scrollbar on self
        self.canvas.configure(
            yscrollcommand=self.vsb.set
        )  # attach scrollbar action to scroll of canvas

        self.vsb.pack(side="right", fill="y")  # pack scrollbar to right of self
        self.canvas.pack(
            side="left", fill="both", expand=True
        )  # pack canvas to left of self and expand to fil
        self.canvas_window = self.canvas.create_window(
            (4, 4),
```

```
            window=self.viewPort,
            anchor="nw",  # add view port frame to canvas
            tags="self.viewPort",
        )

        self.viewPort.bind(
            "<Configure>", self.onFrameConfigure
        )  # bind an event whenever the size of the viewPort frame changes.
        self.canvas.bind(
            "<Configure>", self.onCanvasConfigure
        )  # bind an event whenever the size of the viewPort frame changes.

        self.onFrameConfigure(
            None
        )  # perform an initial stretch on render, otherwise the scroll region has a tiny bord
er until the first resize

    def onFrameConfigure(self, event):
        """Reset the scroll region to encompass the inner frame"""
        self.canvas.configure(
            scrollregion=self.canvas.bbox("all")
        )  # whenever the size of the frame changes, alter the scroll region respectively.

    def onCanvasConfigure(self, event):
        """Reset the canvas window to encompass inner frame when required"""
        canvas_width = event.width
        self.canvas.itemconfig(
            self.canvas_window, width=canvas_width
        )  # whenever the size of the canvas changes alter the window region respectively.


def search_help():
    tk.messagebox.showinfo(
        "Ajuda de Pesquisa",
        """O programa realiza pesquisas com qualquer forma de preenchimento das entradas. Como
 exemplos:\n
a) Para pesquisar compostos com 8 carbonos que têm aroma Doce, digite "C8" no campo "Fórmula"
e "Doce" no campo "Aroma".\n
b) Para pesquisar todos os compostos, simplesmente pressione o botão de pesquisa sem preencher
 qualquer campo.""",
    )


def add_help():
    tk.messagebox.showinfo(
        "Ajuda de Atualização",
        'Para adicionar um novo composto ao banco de dados, é necessário preencher o campo "No
me" e pelo menos um outro campo disponível.',
    )


def about():
    tk.messagebox.showinfo(
        "Sobre",
        """Banco de Dados de Compostos Aromáticos v1.0 (31/03/2020)\n\nO programa foi criado p
or Arthur Adabo de Camargo e Pedro Alvares Eggers sob orientação do Prof. Dr. Moisés Teles dos
 Santos como projeto de conclusão do curso de Engenharia Química da Escola Politécnica da Univ
ersidade de São Paulo.\n
O programa foi escrito em Python, sendo utilizada a biblioteca TKinter para a interface gráfic
a, e em SQLite3 para o banco de dados. Sua finalidade é fornecer facilmente propriedades sobre
 compostos cujos aromas possam ser interessantes para a indústria de cosméticos, bem como perm
itir a pesquisa de tais aromas.""",
    )


if __name__ == "__main__":
    root = tk.Tk()
    menubar = tk.Menu(root)

    help_menu = tk.Menu(menubar, tearoff=0)
```

```
    menubar.add_cascade(label="Ajuda", menu=help_menu)
    help_menu.add_command(label="Pesquisa", command=search_help)
    help_menu.add_command(label="Atualização de Banco de Dados", command=add_help)
    help_menu.add_command(label="Sobre", command=about)
    root.config(menu=menubar)
    root.geometry("800x600+300+300")
    App(root, background="#F0F0F0").pack(side="top", fill="both", expand=True)
    root.minsize(800, 600)
    root.mainloop()
```

9.2 Programa desenvolvido em Python para fazer as queries no banco de dados *(query.py)*

```
"""
Autores:

    Arthur Adabo de Camargo, nº USP: 9834128
    Pedro Alvares Eggers, nº USP: 9833440

Fonte: https://www.sqlitetutorial.net/sqlite-python/create-tables/
"""

import sqlite3
from sqlite3 import Error
import csv
import os

CURR_PATH = os.path.dirname(os.path.abspath(__file__))

def create_connection(db_file):
    """ create a database connection to the SQLite database
        specified by db_file
    :param db_file: database file
    :return: Connection object or None
    """
    conn = None
    try:
        conn = sqlite3.connect(db_file)
        return conn
    except Error as e:
        print(e)

    return conn

def create_table(conn, create_table_sql):

    """ create a table from the create_table_sql statement
    :param conn: Connection object
    :param create_table_sql: a CREATE TABLE statement
    :return:

    """
    try:
        c = conn.cursor()
        c.execute(create_table_sql)
    except Error as e:
        print(e)

def create_db():

    """
    Fonte: https://www.sqlitetutorial.net/sqlite-python/create-tables/

    """

    database = os.path.join(CURR_PATH, "camd_db.db")

    sql_create_projects_table = """ CREATE TABLE IF NOT EXISTS molecule_table (
                                        smiles text NOT NULL,
                                        odour text NOT NULL,
                                        compound_name text PRIMARY KEY,
                                        formula text NOT NULL,
                                        boiling_point text NOT NULL,
                                        melting_point text NOT NULL,
                                        flash_point text NOT NULL,
                                        solubility text NOT NULL,
                                        vapor_pressure text NOT NULL,
                                        density text NOT NULL,
```

```
                                        vapor_density text NOT NULL,
                                        pka text NOT NULL
                                    ); """

    # create a database connection
    conn = create_connection(database)

    # create tables
    if conn is not None:
        # create projects table
        create_table(conn, sql_create_projects_table)
    else:
        print("Error! cannot create the database connection.")

def read_molecule_csv():

    """
    Transforma um arquivo .csv em lista
    """

    with open(os.path.join(CURR_PATH, "to_database.csv"), "r") as f:
        reader = csv.reader(f, delimiter=";")
        table_list = list(map(tuple, reader))

    return table_list

def update_db(values_list):

    database = os.path.join(CURR_PATH, "camd_db.db")
    conn = create_connection(database)
    print(values_list)

    try:
        c = conn.cursor()

        c.executemany(
            """
        INSERT OR REPLACE INTO molecule_table (smiles, odour, compound_name, formula, boiling_
point, melting_point,
        flash_point, solubility, vapor_pressure, density, vapor_density, pka)
        VALUES (?,?,?,?,?,?,?,?,?,?,?,?)
        """,
            values_list,
        )

        conn.commit()

        print("Dados inseridos com sucesso.")

        conn.close()

    except Error as e:
        print(e)

def get_data(
    smiles="",
    compound_name="",
    formula="",
    boiling_point="",
    melting_point="",
    flash_point="",
    solubility="",
    vapor_pressure="",
    density="",
    vapor_density="",
    pka="",
    odour="",
):
    query = f"SELECT * FROM molecule_table WHERE smiles LIKE '%{smiles}%' AND compound_name LI
KE '%{compound_name}%' AND formula LIKE '%{formula}%' AND boiling_point LIKE '%{boiling_point}
%' AND melting_point LIKE '%{melting_point}%' AND flash_point LIKE '%{flash_point}%' AND solub
```

```
ility LIKE '%{solubility}%' AND vapor_pressure LIKE '%{vapor_pressure}%' AND density LIKE '%{d
ensity}%' AND vapor_density LIKE '%{vapor_density}%' AND pka LIKE '%{pka}%' AND odour LIKE '%{
odour}%'"
    database = os.path.join(CURR_PATH, "camd_db.db")
    conn = create_connection(database)
    c = conn.cursor()
    aux = c.execute(query).fetchall()
    conn.close()

    return list(aux)

def get_odours():

    database = os.path.join(CURR_PATH, "camd_db.db")
    conn = create_connection(database)
    c = conn.cursor()

    ODOURS = [""] + sorted(
        list(
            set(
                [
                    x[0].split(",")[0]
                    for x in c.execute(
                        "SELECT DISTINCT odour FROM molecule_table"
                    ).fetchall()
                ]
            )
        )
    )

    return ODOURS

def delete_compound(name):
    database = os.path.join(CURR_PATH, "camd_db.db")
    try:
        conn = create_connection(database)
        c = conn.cursor()
        c.execute(f'DELETE FROM molecule_table WHERE compound_name="{name}";')
        conn.commit()
        conn.close()
        return True
    except Exception:
        return False
```

9.3 Programa para obter os dados de compostos químicos do *PubChem (import_pubchem.py)*

```
"""
Autores:
    Arthur Adabo de Camargo, nº USP: 9834128
    Pedro Alvares Eggers, nº USP: 9833440

"""
import pubchempy as pcp
import csv
import xlsxwriter
import os

CURR_PATH = os.path.dirname(os.path.abspath(__file__))

def read_csv():

    # abre o arquivo
    with open(os.path.join(CURR_PATH, 'molecules.csv'), 'r') as f:
        reader = csv.reader(f, delimiter=';')
        # faz uma lista com ele. Por isso, é diferente da função read_molecule_csv()
        table_list = list(reader)

    return table_list

def import_from_pubchem():

    compounds = read_csv()
    # cria uma planilha no mesmo local do arquivo .py
    workbook = xlsxwriter.Workbook(filename='to_database.xlsx')
    # cria uma aba
    worksheet = workbook.add_worksheet(name='results')
    row = 1

    print('\nEstabelecendo conexão com o PubChem...')
    # para cada composto na tabela
    for comp in compounds:

        # pega dados no pubChem
        results = pcp.get_compounds(comp[0], 'smiles')
        # baixa a imagem de composto
        pcp.download('PNG', os.path.join(CURR_PATH, 'images', comp[0] + '.png'), comp[0], 'smi
les', overwrite=True)

        # para cada resultado, escreve na planilha nova o SMILES, o aroma, o nome IUPAC e a fó
rmula molecular
        for c in results:

            print('\nComposto ' + c.iupac_name)
            worksheet.write(row, 0, comp[0])
            worksheet.write(row, 1, comp[1])
            worksheet.write(row, 2, c.iupac_name)
            worksheet.write(row, 3, c.molecular_formula)

            row += 1

    workbook.close()
    print('Pronto! Compostos Atualizados')

import_from_pubchem
```

9.4 Programa de *Web-Scraping (get_properties.py)*

```
import requests
from selenium import webdriver
import time
import csv

with open('to_database.csv', 'r') as csvfile:
    componds = csv.reader(csvfile, delimiter=';')
    nome_dos_compostos = [x[2] for x in componds]
nome_das_propriedades = ['Odor', 'Boiling-Point', 'Melting-Point', 'Flash-
Point', 'Solubility', 'Vapor-Pressure', 'Density', 'Vapor-Density', 'pKa']

chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument('--headless')

i = 1
deu_ruim = []
with open('/home/pedro/Desktop/Properties.csv', 'w') as f:
    for nome_do_composto in nome_dos_compostos:
        r = requests.get(f'https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/{nome_do_co
mposto}/cids/TXT')
        cid = ''
        if r.status_code != 200:
            print(f'Deu errado pra pegar o cid do composto {nome_do_composto}')
        else:
            cid = r.text.strip()

        if cid:
            properties_line = f'{nome_do_composto}^^^'
            for propriedade in nome_das_propriedades:
                driver = webdriver.Chrome('/home/pedro/Documents/TCC/chromedriver', options=ch
rome_options)
                driver.get(f'https://pubchem.ncbi.nlm.nih.gov/compound/{cid}#section={propried
ade}&fullscreen=true')
                time.sleep(2)
                properties_line += str([x.text.split("\n")[0] for x in driver.find_elements_by
_css_selector(f'#{propriedade} > div.section-content > div.section-content-item')]) + '^^^'
                if propriedade == 'pKa':
                    properties_line += '\n'
                print(properties_line)
                driver.quit()

            print(str(i) + '/' + str(len(nome_dos_compostos)))
            i += 1
            f.write(properties_line)
            print(properties_line)
        else:
            deu_ruim.append(nome_do_composto)
```