chatcof在Jupyter Lab上的实现

利用jupyter中的魔法方法，运行代码以创建COFs查询器，实现在jupyter lab中查询对应的COF合成物合成方案

数据库更新

在上次的数据基础上，补充了一些文献的OOI，以及将OOI链接的格式规范化

DOI的格式将全部变为`https://doi.org/10.1016/j.chroma.2022.463575` 的形式

```python
import traceback
import urllib.parse

import pandas as pd
import pymysql


def clear_str(text):
    """将字符串中多余的引号转换，便于写入数据库"""
    if isinstance(text, str):
        text = text.replace('"', "'")
        text = " ".join([i.strip() for i in text.split()])
        return text
    else:
        return ""


def handle_doi(doi):
    """调整doi的格式"""
    if not doi.strip():
        return doi
    doi = doi[:4].replace("DOI", "").replace("doi", "").replace(":", 
"").replace(": ", "").strip() + doi[4:].strip()
    if "http" not in doi:
        doi = doi.replace("info:doi/", "")
        new_doi = urllib.parse.urljoin("https://doi.org/", doi)
        return new_doi
    else:
        return doi


def update_excel():
    """更新数据库中的文献doi"""
    conn = pymysql.Connect(host="127.0.0.1", port=3306, user='root', 
password='123456', db='chatcof')
    db = conn.cursor()

    df = pd.read_excel("Results.xlsx", sheet_name="synthesis paragraph")
    data_dict = df.to_dict(orient='records')

    for item in data_dict:
        doc = clear_str(item["Literature"])
        if not doc:
            continue
        doi = clear_str(item["DOI"])

        query_sql = f"""SELECT id, doi FROM `cof_documents` WHERE `name` = "
{doc}";"""
        db.execute(query_sql)
        res = db.fetchone()
        new_doi = handle_doi(doi)
        print(new_doi)

        try:
            sql = f"""UPDATE `chatcof`.`cof_documents` SET `doi` = "{new_doi}"
WHERE `id` = {res[0]};"""
            db.execute(sql)
            conn.commit()
        except Exception as e:
            traceback.print_exc()
            conn.rollback()
```

环境配置

安装好Anconda后，首先要安装依赖环境

pandas Anconda自带，不需要额外安装，只需要装pymysql就行

打开Anconda Prompt输入以下指令：

```
pip install pymysql
```

代码实现

打开Ancoda Navigator，选择Jupyter Lab，点击Launch按钮，稍候会弹出的浏览器窗口

在Jupyter Lab界面里，创建一个文件夹chatcof，做为本项目的根目录

然后创建`database.py`文件：

```
import re

import pymysql


class SynthesisDatabase:
    def __init__(self):
        self.conn = pymysql.Connect(host="127.0.0.1", port=3306, user='root',
password='123456', db='chatcof')
        self.db = self.conn.cursor()
        self.temp_re = re.compile(r'(\d+)\s*([°℃Cc]|度|摄氏度)')
        self.time_re = re.compile(r'(\d+)\s*(h|hour|hours|day|days|小时|天)')

    def __del__(self):
        self.db.close()
        self.conn.close()

    def query_complex_names(self, keyword) -> tuple:
        """
        Query all compound names
        :return:
        """
        sql = f"""SELECT `id`, `name` FROM `chatcof`.`cof_complex_names` WHERE
`name` LIKE '%{keyword}%' """
        self.db.execute(sql)
        res = self.db.fetchall()
        return res

    def query_paragraphs(self, name_id) -> list:
        """
        Query paragraphs based on id
        :param name_id:
        :return:
        """
        # Query the composition scheme paragraph based on the ID
        sql = f"""SELECT par.content, doc.doi FROM cof_paragraphs AS par
                    INNER JOIN cof_documents AS doc
                    ON (par.doc_id=doc.id)
                    WHERE par.name_id = "{name_id}";"""
        self.db.execute(sql)
        res = self.db.fetchall()
        return res

    def extract_subject(self, text: str) -> list:
        """
        Extract the subject
        :param text:
        :return:
        """
        punctuation = ["?", "？"]
        for punct in punctuation:
            text = text.replace(punct, "")
        if re.findall(r"[\u4e00-\u9fa5]{1,}", text):
            text_list = re.sub(r"[\u4e00-\u9fa5]{1,}", " ", text).split()
            return text_list
        else:
            stopwords = ["how", " to ", "synthesis", "synthesize"]
            for word in stopwords:
                text = text.replace(word, " ")
                text = text.replace(word.title(), " ")
            if not text.strip():
                return []
            text_list = text.split()
        return text_list

    def com_query_paragraphs(self, text_list):
        """
        There are 2 synthesis schemes for comprehensive query of synthesis:
            - Query the composition ID by using the composition > search the
corresponding paragraph
            - Query the name of the composition and fuzzily search the
corresponding paragraph
        :param text_list:
        :return:
        """
        if not text_list:
            return []
        result = []
        for word in text_list:
            data_tuple = self.query_complex_names(word)
            if data_tuple:
                for data in data_tuple:
                    res = self.query_paragraphs(data[0])
                    value = [{
                        "name": data[1],  # The name of the actual compound
                        "method": i[0],  # Synthesis method
                        "doi": i[1]
                    } for i in res]
                    result.extend(value)
        return result

    def search_synthesis(self, query):
        """
        Search for a synthesis scheme
        :param query: Search keywords
        :return:
        """
        text_list = self.extract_subject(query)
        found_syntheses = self.com_query_paragraphs(text_list)
        if found_syntheses:
            sorted_syntheses = sorted(found_syntheses, key=lambda d:
similarity_check(query, d['name']), reverse=False)[:5]
            return {
                'found': True,
                'syntheses': sorted_syntheses
            }
        else:
            return {
                'found': False,
                'syntheses': []
            }

    def search(self, keyword):
        """
        Search for keywords in the database for fuzzy matching based on
temperature and time conditions
        :param keyword: Input text containing temperature and/or time conditions
        :return: Dictionary containing search results
        """
        # Extract temperature (supports ℃, °C, C formats)
        temp_match = self.temp_re.search(keyword)
        temperature = temp_match.group(1) if temp_match else None

        # Extract time (supports h, hours, day, days formats)
        time_match = re.search(r'(\d+)\s*(h|hour|hours|day|days|小时|天)',
keyword)
        time_value = time_match.group(1) if time_match else None
        time_unit = time_match.group(2) if time_match else None
        is_hours = 'h' in time_unit or 'hour' in time_unit or '小时' in time_unit
if time_match else None

        # Convert hours to days if necessary (72h = 3 days)
        if time_value and time_unit and is_hours:
            days_value = str(int(int(time_value) / 24))  # Convert to days
            hours_value = time_value  # Keep original hours value
        # Build SQL query based on conditions
        conditions = []
        params = []

        if temperature:
            # Add temperature match condition with variations
            conditions.append("""
                (par.content REGEXP %s OR
                 par.content REGEXP %s OR
                 par.content REGEXP %s OR
                 par.content REGEXP %s)
            """)
            params.extend([
                f"[^0-9]{temperature}[°℃C]",
                f"[^0-9]{temperature}c",
                f"[^0-9]{temperature}C",
                f"[^0-9]{temperature}度"
            ])

        if time_value:
            # Search for both days and hours format
            if time_unit and is_hours:
                # For hours input, search for exact hour match and corresponding
days
                conditions.append("""
                    (par.content LIKE %s OR par.content LIKE %s OR
                     par.content LIKE %s OR par.content LIKE %s OR
                     par.content LIKE %s OR par.content LIKE %s OR
                     par.content LIKE %s)
                """)
                params.extend([
                    f"%{hours_value}h%",
                    f"%{hours_value} hour%",
                    f"%{hours_value} hours%",
                    f"%{hours_value}小时%",
                    f"%{days_value} day%",
                    f"%{days_value} days%",
                    f"%{days_value}d%"
                ])
            else:
                # For days input, search for exact day match
                conditions.append("(par.content LIKE %s OR par.content LIKE %s
OR par.content LIKE %s OR par.content LIKE %s)")
                params.extend([
                    f"%{time_value} day%",
                    f"%{time_value} days%",
                    f"%{time_value}d%",
                    f"%{time_value}天%"
                ])

        if not conditions:
            return {
                'found': False,
                'syntheses': []
            }

        # Query paragraphs with the specified conditions
        sql = f"""
            SELECT DISTINCT cn.name, par.content, doc.doi
            FROM cof_paragraphs AS par
            INNER JOIN cof_documents AS doc ON (par.doc_id=doc.id)
            INNER JOIN cof_complex_names AS cn ON (par.name_id=cn.id)
            WHERE {' AND '.join(conditions)}
        """

        try:
            self.db.execute(sql, params)
            results = self.db.fetchall()

            if not results:
                return {
                    'found': False,
                    'syntheses': []
                }

            # Format results similar to search_synthesis
            syntheses = [{
                'name': result[0],
                'method': result[1],
                'doi': result[2]
            } for result in results]

            # Sort and limit results similar to search_synthesis
            sorted_syntheses = sorted(syntheses, key=lambda d:
similarity_check(keyword, d['name']), reverse=False)[:5]

            return {
                'found': True,
                'syntheses': sorted_syntheses
            }

        except Exception as e:
            print(f"Database query error: {str(e)}")
            return {
                'found': False,
                'syntheses': []
            }


def similarity_check(keyword, text):
    """
    The similarity of two strings is calculated based on the length of the string
difference
    :param keyword: The keyword of the composition entered by the user
    :param text: The name of the composition in the database
    :return:
    """
    res = text.upper().replace(keyword.upper(), "")
    if "COF" not in keyword:
        res = res.replace("-COF", "").replace("COF", "")
    return len(res.strip())


def take_star_2(keyword, cof_name):
    diff_length = similarity_check(keyword, cof_name)
    if diff_length == 0:
        star = "★★★★★"
    elif diff_length <= 3:
        star = "★★★★☆"
    elif diff_length <= 6:
        star = "★★★☆☆"
    elif diff_length <= 9:
        star = "★★☆☆☆"
    else:
        star = "★☆☆☆☆"

    return star


def take_star_1(index):
    return (5 - index) * "★" + index * "☆"


def process_synthesis_data(keyword, data, sort):
    if data['found'] and data['syntheses']:
        message = ''
        for index, synthesis in enumerate(data.get('syntheses', [])):
            if sort == 1:
                star = take_star_1(index)
            else:
                star = take_star_2(keyword, synthesis['name'])

            message += f"Synthesis Method {index + 1}:\n\n"  # Add title
            message += f"[Synthesis Steps] {star}\n"  # Process synthesis steps,
ensure each step on new line

            steps = synthesis.get('method', '').split('。')
            for step in steps:
                step = step.strip()
                if step:
                    message += f"{step}。\n"

            # Add DOI reference with check
            message += f"\nFor more details, see:\n{synthesis.get('doi', 'DOI
not found for this paper')}\n"

            # Add separator if not last synthesis
            if index < len(data['syntheses']) - 1:
                message += '\n\n----------------------------------------\n\n'

        print(message)
    else:
        web_of_science_msg = ("No relevant synthesis methods found.\n\n"
                              "You can search related literature on Web of
Science:\n"
                              "https://www.webofscience.com/wos")
        print(web_of_science_msg)


def run(keyword, sort=2):
    """
    Search synthesis methods by compound name or conditions (temperature/time)
    :param keyword: Can be compound name (e.g., "TAPB") or conditions (e.g.,
"120°C 72h")
    :param sort: The display mode of the star rating in the printed message,
            1- mandatory press 5★-1★ display;
            2- Display according to the actual match
    :return:
    """
    # Check if the input contains temperature or time patterns
    temp_pattern = sd.temp_re.search(keyword)
    time_pattern = sd.time_re.search(keyword)

    if temp_pattern or time_pattern:
        # Use condition-based search
        res = sd.search(keyword)
    else:
        # Use compound name search
        res = sd.search_synthesis(keyword)

    process_synthesis_data(keyword, res, sort)


if __name__ == '__main__':
    """
    How to Use:
    1. Open Jupyter Lab, open the folder of the file, or copy the file to the
project folder of Jupyter Lab
    2. Create a notebook file and enter %run database.py

    Examples:
    - Search by compound name: run("TAPB")
    - Search by conditions: run("120°C 72h")
    """
    sd = SynthesisDatabase()
    print("-Load done-")
    print('You can use the `run()` to search synthesis schemes:\n'
          '1. Search by compound name, e.g.: run("TAPB")\n'
          '2. Search by conditions, e.g.: run("120°C 72h")')
```

这段代码中定义了一个cof的查询器，运行它会提供给一个`run()`函数，我们可以借此来查询cof的合成方案

运行说明

在Jupyter Lab界面的chatcof文件夹里，创建一个Notebook文件

然后在第一行输入：

```
%run database.py
```

> 这一步是让jupyter运行咱们的cof查询器，并将run函数加载到内存中，便于之后随时调用

然后就可以在第二行运行run函数来获取对应的cof合成物了

使用示例

程序支持两种星级展示方式（以当前最多显示5个来举例）

1.获取TAPB的合成方案，以第一种模式展示星级

```
run("tapb", sort=1)
```

固定展示5★-1★，合成方案的显示顺序是按照**对应化合物名称**与**用户指定的化合物名称**的**相似度**来排序

2.获取TAPB的合成方案，以第二种模式展示星级

```
run("tapb")
# 或者
run("tapb", sort=2)
```

按照**对应化合物名称**与**用户指定的化合物名称**的**相似程度**，来制定一套打分机制，相似度越高星级越高

打分机制为：

1. ★★★★★-与目标化合物相差字符数 = 0
2. ★★★★☆-与目标化合物相差字符数 <= 3
3. ★★★☆☆-与目标化合物相差字符数 <= 6
4. ★★☆☆☆-与目标化合物相差字符数 <= 9
5. ★☆☆☆☆-与目标化合物相差字符数 > 9

目前程序默认为第二种模式

3.基于温度和时间条件搜索合成方案

```
# 搜索特定温度的合成方案
run("120℃")

# 搜索特定时间的合成方案
run("72h")          # 按小时搜索
run("3 days")       # 按天数搜索

# 组合搜索温度和时间条件
run("120℃ 72h")     # 同时搜索120℃和72小时的合成方案
```

搜索条件支持的格式：

- 温度：支持 ℃、°C、C 等格式，如：120℃、120°C、120C
- 时间：
  - 小时：支持 h、hour、hours 格式，如：72h、72 hours
  - 天数：支持 day、days 格式，如：3 days、3d

所有搜索结果都支持两种星级显示模式：

1. sort=1：固定按照5★到1★显示
2. sort=2（默认）：根据匹配度显示星级

- ★★★★★：完全匹配
- ★★★★☆：相差字符数 ≤ 3
- ★★★☆☆：相差字符数 ≤ 6
- ★★☆☆☆：相差字符数 ≤ 9

- ★☆☆☆☆：相差字符数 > 9

如果没有找到相关的合成方案，程序会提示访问 Web of Science 搜索相关文献。