**数据库系统project报告**

2022-2023学年第2学期（CST21118）

|  |  |
| --- | --- |
| 数据库系统project任务书 | |
| 名称 | 数据库SQL执行器设计与模拟实现 |
| 类型 | □验证性 □设计性 综合性 |
| 内容 | 针对关系型数据库SQL语句执行，进行需求分析，并设计一个简单的数据库系统SQL执行器模块，根据设计模拟实现SQL基本操作等，主要实现：数据库创建、表格创建、数据添加、删除、更新、查询等操作。  模拟实现采用：python或者Java实现具体功能，设计中若有数据的存储可以使用文本文件或者excel文件，SQL语句执行采用函数实现。 |
| 要求 | （1）设计方案要合理；  （2）能基于该SQL执行器模块完成SQL的模拟实现；  （3）设计方案有一定的效率分析。 |
| 任务时间 | 2023年4月17日至2023年5月20日 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 小组成员 | | | | | | |
| 学号 | | 学号 | | 学号 | | 学号 |
|  | | 姓名 | | 姓名 | | 姓名 |
| 项目评分表 | | | | | | |
| 序号 | 评分项 | | 分值 | | 得分 | |
| 1 | 需求分析 | | 3分 | |  | |
| 2 | 综合设计与实现 | | 4分 | |  | |
| 3 | 团队协作 | | 3分 | |  | |
| 项目总得分： | | | | | | |

课程项目评分标准（总分10分）

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 评分项目 | 完成情况 | 得分 |
| 1 | 需求分析 | 分析合理 | 3分 |
| 分析较合理 | 2分 |
| 分析不合理 | 1分 |
| 分析完全错误 | 0分 |
| 2 | 综合设计与实现 | 设计完整，设计合理，工具使用熟练 | 4分 |
| 设计较完整，设计合理，工具使用较熟练 | 3分 |
| 设计较完整，设计较合理，工具使用较熟练 | 2分 |
| 设计较完整，设计不合理，工具使用不熟练 | 1分 |
| 抄袭、被抄袭 | 0分 |
| 3 | 团队协作 | 有团队，分工合理，密切协作 | 3分 |
| 有团队，分工合理，有一定协作 | 2分 |
| 有团队，分工不合理，无协作 | 1分 |
| 无团队，无协作 | 0分 |

**1.需求分析**

针对项目要求，我们小组讨论需求分析，明确数据存储管理的功能需求以便后续的分析。我们考虑到用户对数据库的功能使用要求，对于用户，设计了创建数据库、创建表、增加（插入）数据、删除数据、更新数据、查询数据、以及用户帮助功能。在本数据存储管理器中，设计用户用例多项，并进行组织、记录或修改。每个用例分析如下：

·**创建数据库**：create\_database : CREATE DATABASE demo

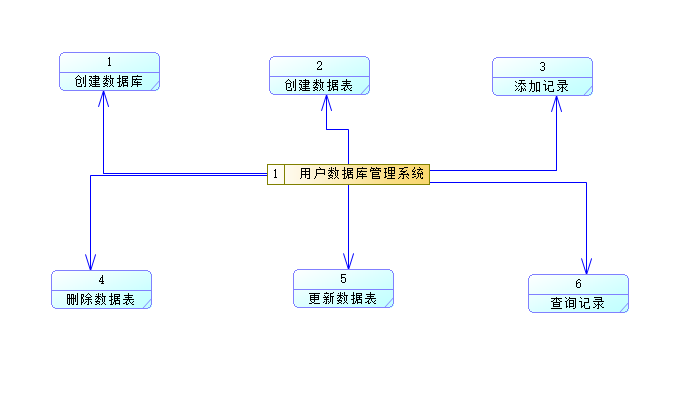
·**创建数据表：**create\_table : CREATE TABLE demo.demo\_table (name varchar(50),age int,gender varchar(1)) ;

·**添加记录：**insert : INSERT INTO demo.demo\_table VALUES (张三,21,男) ;

·**删除数据表**delete : DELETE FROM demo.demo\_table WHERE name=张三 ;

**·更新数据表**update : UPDATE demo.demo\_table SET name=李四,age=18 WHERE name=张三 ;

**·查询记录** query : ELECT name,age,gender FROM demo.demo\_table WHERE name=张三 and age=21 ;或者 SELECT name,age,gender FROM demo.demo\_table ;



**2.综合设计与实现**

**2.1数据库管理模块**

2.1.1创建数据库

创建一个新的数据库文件夹，用于存储数据库相关的表格和数据。当用户登录进入数据库存储管理器后，可以使用create\_database语句创建数据库。我们直接在当前路径(当前.py文件所在路径)下创建数据库。

2.1.2创建表格

在指定的数据库中创建一个新的表格，使用create\_table语句，包括表格名称、字段名称和数据类型等信息。

**2.2数据操作模块**

2.2.1数据插入：将数据插入到指定表格中，确保数据的完整性和一致性。

2.2.2数据查询：支持基本的SELECT语句，包括查询特定字段、过滤条件、排序和限制等。

2.2.3.数据更新：更新表格中满足条件的数据，确保数据的一致性。

2.2.4数据删除：删除表格中满足条件的数据，确保数据的完整性。

2.2.5数据索引：支持索引的创建和利用，以提高查询效率。

**2.3 SQL语句解析和执行模块**

2.3.1SQL解析：解析输入的SQL语句，提取关键信息，包括操作类型、表格名称、字段名称、过滤条件等。 其中以下是输入SQL语句的规则：

一些说明：

    1、文件统一编码为：UTF-8

    2、sql语句请严格符合以下例子，关键词请大写，其他小写，空格严格遵守以下例子：

            create\_database :   CREATE DATABASE demo ;

            create\_table :      CREATE TABLE demo.demo\_table (name varchar(50),age int,gender varchar(1)) ;

            insert :            INSERT INTO demo.demo\_table VALUES (张三,21,男) ;

            delete :            DELETE FROM demo.demo\_table WHERE name=张三 ;

            update :            UPDATE demo.demo\_table SET name=李四,age=18 WHERE name=张三 ;

            query :             SELECT name,age,gender FROM demo.demo\_table WHERE name=张三 and age=21 ;

                          或者   SELECT name,age,gender FROM demo.demo\_table ;  # 查询全部结果

2.3.2 SQL执行计划：根据解析得到的SQL语句，生成相应的执行计划，确定执行顺序和方式。

2.3.3 SQL执行：根据执行计划，调用数据库管理模块和数据操作模块，执行相应的操作。

**3.代码实现思路**

**3.1 创建数据库**

def create\_database(sql):

    """

    创建数据库，直接在当前路径(当前.py文件所在路径)下创建数据库

    :param sql: CREATE DATABASE demo ;

    :return: None

    """

    database\_name = sql[16:-1].strip()

    if not os.path.exists(database\_name):

        os.mkdir(database\_name)

        print(f'{database\_name}数据库成功创建！')

    else:

        print(f'{database\_name}数据库已经存在！')  # 数据库不能重名

函数名为 create\_database，接受一个参数sql表示SQL语句。此函数的作用是根据输入的SQL语句，在当前路径下创建一个数据库。

首先从 SQL 语句中解析出数据库名称（在这里假设 SQL 语句格式为 CREATE DATABASE demo ;，因此需要去掉字符串前后的空格，并提取其中的 demo）。然后检查当前路径下是否存在同名的文件夹（即数据库），如果不存在，则创建一个新的文件夹（即新建一个数据库）。如果已经存在同名的文件夹，则输出提示信息并退出函数。最后根据操作结果输出相应的提示信息。

**3.2 创建表**

def create\_table(sql):

    """

    新建一个表

    :param sql: CREATE TABLE demo.demo\_table (name varchar(50),age int,gender varchar(1)) ;

    :return: None

    """

    index\_Dot, index\_bracket = sql.index('.'), sql.index('(')

    attributes\_all = sql[index\_bracket + 1:-3]

    attributes\_respect = attributes\_all.split(',')

    attributes\_list = []

    for attributes in attributes\_respect:

        idx = attributes.index(' ')

        attributes\_list.append(attributes[:idx])

    database\_name = sql[13:index\_Dot + 1]

    table\_name = sql[index\_Dot + 1: index\_bracket - 1]

    table\_path = './' + database\_name + '/' + table\_name + '.csv'

    if os.path.exists(table\_path):

        print(f"表{table\_name}已经存在！")

    else:

        with open(table\_path, 'w', newline='', encoding='UTF-8') as file:

            writer = csv.writer(file)

            writer.writerow(attributes\_list)

            print(f"表{table\_name}创建成功！")

这段代码实现了一个函数 create\_table(sql)，用于创建一个新的表。该函数的输入参数 sql 是一个 SQL 语句字符串。

在主体部分，代码首先通过 index() 方法获取 SQL 语句中点号和左括号的位置，然后根据这些信息提取出数据库名、表名和列名等必要的信息。接着，代码检查表是否已经存在，如果不存在，则创建一个 CSV 文件并写入该表的列名。

**3.3 表的功能**

3.3.1 插入数据

def insert(sql):

    """

    向表中插入数据

    :param sql: INSERT INTO demo.demo\_table VALUES (张三,21,男) ;

    :return: None

    """

    idx\_dot, idx\_bracket = sql.index('.'), sql.index('(')

    database = sql[12:idx\_dot]  # 获取数据库名

    if os.path.exists(database):

        table\_name = sql[idx\_dot + 1:idx\_bracket - 8]  # 获取表名

        table\_path = './' + database + '/' + table\_name + '.csv'

        if os.path.exists(table\_path):

            values = sql[idx\_bracket + 1:-3].split(',')

            with open(table\_path, 'a', newline='', encoding='UTF-8') as file:

                writer = csv.writer(file)

                writer.writerow(values)  # 将数据写入最后一行

                print("数据插入成功！")

        else:

            print("数据库" + database + "不存在表" + table\_name + "!")

    else:

        print("不存在数据库" + database + "!")

实现了基于SQL语句的插入数据功能。该函数包含一个参数 sql，用于传入待执行的SQL语句。函数首先从SQL语句中提取出要插入的数据库、表名以及插入的数据，并检查对应的CSV文件是否存在。如果CSV文件不存在，则输出相应的错误信息；否则，将指定的数据写入到CSV文件中。

具体来说，该函数使用csv模块打开CSV文件，并使用writerow方法将待插入的数据一行一行地写入CSV文件的最后。如果文件不存在，则创建该文件并写入数据。

如果SQL语句中指定的数据库或表不存在，则输出相应的错误信息。否则，输出插入成功的提示信息。

3.3.2 删除记录

def delete(sql):

    """

    数据库删除记录

    注意：代码中delete\_row应修改为=where(sql),返回的是行数，在这里，csv文件属性名行为第0行,数据是第1行

    :param sql: DELETE FROM demo.demo\_table WHERE name=张三 ;

    :return: None

    """

    idx\_dot = sql.index('.')

    database = sql[12:idx\_dot]  # 获取数据库名

    table\_name = sql[idx\_dot+1:].split(' WHERE')[0]  # 获取表名

    if os.path.exists(database):

        table\_path = './' + database + '/' + table\_name + '.csv'

        if os.path.exists(table\_path):

            delete\_row = where(sql)

            if not delete\_row:  # 删除的行不存在

                print('数据不存在！')

                return None

            delete\_rowNew = [num - 1 for num in delete\_row]  # 适应python的编号

            data = pd.read\_csv(table\_path, encoding='UTF-8')  # 用pandas读入数据

            data.drop(delete\_rowNew, inplace=True)  # 输出指定delete\_row中对应的行

            data.dropna(inplace=True)  # 删除空行

            data.to\_csv(table\_path, index=False, encoding='UTF-8')  # 覆盖写

            print("数据删除成功！")

        else:

            print("数据库" + database + "不存在表" + table\_name + "!")

    else:

        print("不存在数据库" + database + "!")

实现了基于SQL语句的删除数据功能。该函数包含一个参数 sql，用于传入待执行的SQL语句。函数首先从SQL语句中提取出要删除的数据库、表名以及删除条件，并读取对应的CSV文件。然后根据WHERE子句筛选出要删除的行，逐一将其从读取的数据中删除，并将更新后的结果再次写入到CSV文件中。

具体来说，该函数使用pandas库读取CSV文件，按照WHERE子句匹配需要删除的行，并使用drop方法将这些行从读取的数据中删除。随后，调用dropna方法删除空白行，并使用to\_csv方法把更新后的结果保存回CSV文件中。

如果SQL语句中指定的数据库、表或行不存在，则输出相应的错误信息。否则，输出删除成功的提示信息。

3.3.3 更新数据库

def update(sql):

    """

    update更新数据库语句

    :param sql: UPDATE demo.demo\_table SET name=李四,age=18 WHERE name=张三 ;

    :return: None

    """

    idx\_dot = sql.index('.')

    database = sql[7:idx\_dot]  # 获取数据库名

    table\_name = sql[idx\_dot+1:].split(' SET')[0]  # 获取表名

    if os.path.exists(database):

        table\_path = './' + database + '/' + table\_name + '.csv'

        if os.path.exists(table\_path):

            update\_row = where(sql)

            if not update\_row:  # 更新的行不存在

                print('数据不存在！')

                return None

            condition = re.findall(r'SET(.\*?)WHERE', sql)[0].strip()

            conditions = condition.split(',')

            dict = {item.split('=')[0]: item.split('=')[1] for item in conditions}

            data = pd.read\_csv(table\_path, encoding='UTF-8')

            for i in update\_row:

                for key, value in dict.items():

                    data.loc[i - 1, key] = value

            data.to\_csv(table\_path, index=False,encoding='UTF-8')

            print("数据更新成功！")

        else:

            print("数据库" + database + "不存在表" + table\_name + "!")

    else:

        print("不存在数据库" + database + "!")

函数首先从SQL语句中提取出要更新的数据库、表名以及更新条件，并读取对应的CSV文件。然后根据WHERE子句筛选出要更新的行，逐一更新行中指定的列的值，并将更新后的结果再次写入到CSV文件中。

具体来说，该函数使用pandas库读取CSV文件，按照WHERE子句匹配需要更新的行，并使用字典存储SET子句中指定的要更新的列的值。随后，遍历需要更新的行和指定的列，将对应位置上的值进行修改，最后调用to\_csv方法把更新后的结果保存回CSV文件中。

如果SQL语句中指定的数据库、表或行不存在，则输出相应的错误信息。否则，输出更新成功的提示信息。

3.3.4 查询数据库

def query(sql):

    """

    查询语句

    :param sql: SELECT name,age,gender FROM demo.demo\_table WHERE name=张三 and age=21 ;

        或者     SELECT name,age,gender FROM demo.demo\_table ;  # 查询全部结果

    :return:  None

    """

    # attribute : 列表，这条语句就是：[name, age, grade]

    attribute = sql.split('FROM')[0].split('SELECT')[1].strip().split(',')

    # database\_name: 数据库的名字  table\_name : 表的名字

    if 'WHERE' in sql:

        temp = sql.split('FROM')[1].split('WHERE')[0].strip().split(',')

    else:

        temp = sql.split('FROM')[1].split(';')[0].strip().split(',')

    # 处理多表联查，将联查的结果放在temp文件下，同时修改sql语句，方便where查询处理

    path = '.\\temp\\temp.csv'

    if not os.path.exists('.\\temp'):  # 创建临时的文件夹

        os.makedirs('.\\temp')

    fp = open(path, 'w')  # 相当于清空temp文件 或者 创建temp文件

    fp.close()

    columns = []  # 多表联查的属性

    content = [[]]  # 多表联查的内容

    for s in temp:

        tp = s.split('.')

        database\_name = tp[0]

        table\_name = tp[1]

        if not os.path.exists(database\_name):

            print(f"{database\_name}数据库不存在！")

            return None

        if not os.path.exists(f".\\{database\_name}\\{table\_name}.csv"):

            print(f"{table\_name}表不存在！")

            return None

        path = '.\\' + database\_name + '\\' + table\_name + '.csv'

        data = pd.read\_csv(path, encoding='UTF-8')

        columns += list(data.columns)

        content\_temp = []

        for x in data.values:

            x = list(x)

            for t in content:

                if t:

                    content\_temp.append(t + x)

                else:

                    content\_temp.append(x)

        content = content\_temp

    df = pd.DataFrame(content, columns=columns)  # 准备写入temp文件，方便where处理语句

    df.to\_csv('.\\temp\\temp.csv', index=False, encoding='UTF-8')  # index=False，防止多加一列作为索引

    # 处理列

    columns = set(columns)

    for x in attribute:

        if x not in columns:

            print(f"不存在{x}属性!")

            return None

    # 查询到的数据行

    target\_hang = []

    tag = False

    if 'WHERE' not in sql:

        tag = True

    else:

        sql = sql.replace(sql.split('FROM')[1].split('WHERE')[0].strip(), 'temp.temp')  # 修改sql语句，方便where查询处理

        target\_hang = [0] + where(sql)

    print('===================================================================')

    if not tag:

        data = pd.read\_csv('.\\temp\\temp.csv', skiprows=lambda x: x not in target\_hang, usecols=attribute, encoding='UTF-8')

    else:

        data = pd.read\_csv('.\\temp\\temp.csv', usecols=attribute, encoding='UTF-8')

    if not data.values.size:

        print("无符合条件的查询结果！")

    else:

        print(data)

    print('===================================================================')

    return None

实现了基于SQL语句的查询功能。该函数包含一个参数 sql，用于传入待执行的SQL语句。函数首先从SQL语句中提取出需要查询的列名，以及从哪个数据库、哪张表中查询数据。如果SQL语句中包含WHERE子句，则按照WHERE子句对查询结果进行过滤；否则返回全部结果。

函数中还有一些处理多表联查的逻辑，具体来说是将联查的结果放在临时文件中，并修改SQL语句，方便WHERE查询处理。最后，函数使用pandas库读取CSV文件，将查询结果输出到控制台。

3.4 查看索引

def helper(path, ll):

    """

    where函数的辅助函数，用来查找符合条件的行号

    :param path: 文件路径

    :param ll: sql以空格作为分隔符产生的列表，相当于是sql语句的信息

    :return: 返回符合条件的行号的列表，行号从1开始

    """

    rdata = pd.read\_csv(path, encoding='UTF-8')

    cc = None

    for i, content in enumerate(ll):

        if content == 'WHERE':

            cc = ll[i + 1:]

            break

    ans = set()

    for content in cc:

        if content in ('and', 'or'):

            continue

        if '=' in content:

            col, val = content.split('=')

            q = []

            row = 0

            for data in rdata[col]:

                row += 1

                if str(data) == val:

                    q.append(row)

            if len(ans) == 0:

                ans = set(q)

            else:

                if 'and' in cc:

                    ans = ans & set(q)  # 集合的交运算，and

                elif 'or' in cc:

                    ans = ans | set(q)  # 集合的或运算，or

        elif '>' in content:

            col, val = content.split('>')

            q = []

            row = 0

            for data in rdata[col]:

                row += 1

                if data > int(val):

                    q.append(row)

            if len(ans) == 0:

                ans = set(q)

            else:

                if 'and' in cc:

                    ans = ans & set(q)

                elif 'or' in cc:

                    ans = ans | set(q)

        elif '<' in content:

            col, val = content.split('<')

            q = []

            row = 0

            for data in rdata[col]:

                row += 1

                if data < int(val):

                    q.append(row)

            if len(ans) == 0:

                ans = set(q)

            else:

                if 'and' in cc:

                    ans = ans & set(q)

                elif 'or' in cc:

                    ans = ans | set(q)

    return list(ans)

def where(sql):

    """

    通过sql语句返回查询到的数据索引项列表

    :param sql:string

    :return:list of index where fetch the limitation

    """

    ll = sql.split(' ')

    if ll[0] == "UPDATE":

        path = "." + "".join([("\\" + i) for i in ll[1].split('.')]) + ".csv"

        return helper(path, ll)

    elif ll[0] == "SELECT":

        path = "." + "".join([("\\" + i) for i in ll[3].split('.')]) + ".csv"

        return helper(path, ll)

    elif ll[0] == "DELETE":

        path = "." + "".join([("\\" + i) for i in ll[2].split('.')]) + ".csv"

        return helper(path, ll)

    else:

        return []

where函数的辅助函数，用来查找符合条件的行号。

该函数包含两个参数，path表示要查询的CSV文件路径，ll是sql语句以空格作为分隔符产生的列表，相当于SQL语句的信息。在函数中，先使用pandas库读取对应的CSV文件。然后，使用循环遍历查询条件，根据等于、大于、小于等不同的操作符查找符合条件的行号，并将其添加到集合变量ans中。最后，将集合变量转化为列表返回。

**5.总结**

本次数据库存储管理器的设计与实现，我们通过分头查阅文献，多次的小组讨论，对该数据存储管理器进行需求分析，用例建模，理解功能。我们设计了数据库并分别管理数据库和数据表，通过对这两张表中的操作来实现数据库创建、数据表创建、数据添加、删除等操作。

这个项目的开发过程非常有价值，我们采取了记录与分次讨论的方式来保证每次会议的高效性和有效性。这个项目不仅提高了我们的合作和沟通能力，也让我们从数据库管理系统设计的角度去思考问题并进行代码实现，从而更深入地理解数据库的原理。这种思考方式和实践方法是计算机科学和技术人才必须掌握的工作。