苏州大学实验报告

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 院系 | 计算机学院 | | 年级专业 | | 21计科 | | 姓名 | 方浩楠 | 学号 | 2127405048 |
| 课程名称 | | 编译原理课程实践 | | | | | | | 成绩 |  |
| 指导教师 | | 王中卿 | | 同组实验者 | | 无 | | 实验日期 | 2023.10.30 | |

|  |  |
| --- | --- |
| 实 验 名 称 | SQL查询语句解析 |

1. 实验目的

熟悉并理解SQL查询语言的基本结构和语法规则。

通过编写和扩展代码，深入理解SQL解析程序的工作原理。

利用编写的SQL查询解析程序，实现对CSV文件中数据的查询和分析，提高编程和数据处理能力

1. 实验内容

详细阅读并理解项目中sql\_example(在本实验中为NodeCreation/SQLParser.py和Analysis/SQLAnalysis.py)模块的源代码，理解其解析SQL语句和执行查询的逻辑。

扩展现有的SQL解析程序，增加缺失的SQL语法元素，使其能够完全适应SQL语言的查询语句。

读取student.csv文件，利用编写的SQL查询语言解析程序，构建相应的SQL语句，并完成查询：

1. 实验步骤和结果

本项目的所有 python 代码均根据 Google Style Guide 给出了 docstring 和注释,便于阅读和理解.

同时给出了 readme.md 用来解释项目,并且给出项目的运行方式,同时便于快速理解项目的结构和功能

项目结构:

**|-- AST/**

**| |-- Node.py # 该模块定义了一个用于表示树结构节点的类 Node**

**|-- NodeCreation/**

**| |-- SQLParser.py # 该模块通过 PLY 解析 SQL 查询，并创建对应的抽象语法树 (AST)**

**|-- Analysis/**

**| |-- SQLAnalysis.py # 该模块提供了对抽象语法树 (AST) 中的 SQL 查询进行分析和执行的功能**

**|-- tests/**

**| |-- test\_Node.py # 测试AST/Node.py是否正常生成AST**

**| |-- test\_SQLParser.py # 测试NodeCreation/SQLParser.py根据SQL生成的AST**

**| |-- test\_SQLAnalysis.py # 测试对SQL的分析**

**|-- data/**

**| |-- student.csv # 学生信息**

**|-- run.py # 顶级脚本，用于运行项目**

**|-- README.md # readme文件,用来项目概述和使用说明**

**|-- requirements.txt # 用来说明项目的依赖项**

项目依赖:

**pytest~=7.4.32**

**numpy~=1.26.13**

**pandas~=2.1.24**

**ply~=3.11**

项目运行方式:

在项目根文件夹运行

**pip install -r requirements.txt**

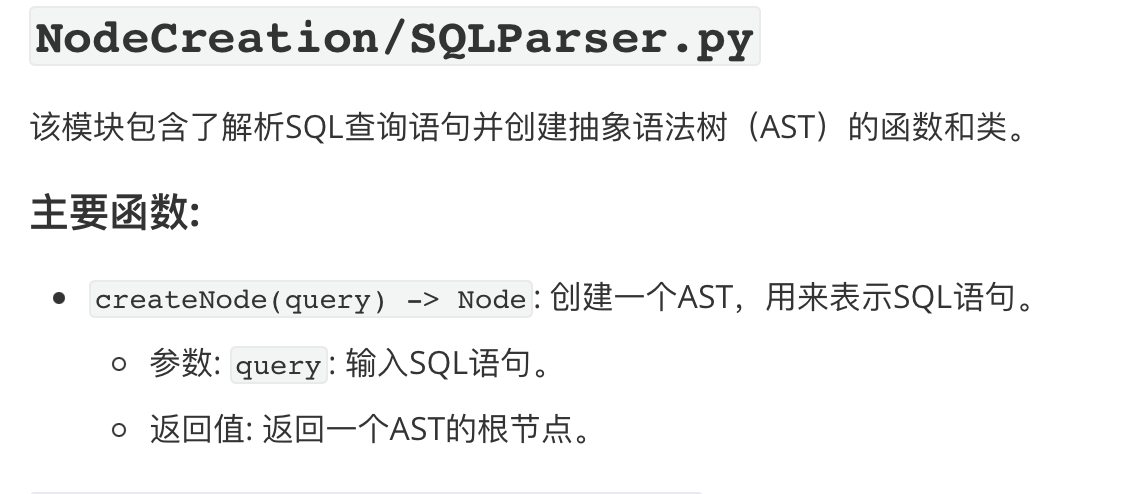
然后运行**run.py**

**python run.py**

这将输出分析结果

模块和接口的说明:







SQLParser.py中的token:

reserved = {

'SELECT': 'SELECT',

'FROM': 'FROM',

'WHERE': 'WHERE',

'ORDER': 'ORDER',

'BY': 'BY',

'AND': 'AND',

'OR': 'OR',

'AVG': 'AVG',

'BETWEEN': 'BETWEEN',

'IN': 'IN',

'SUM': 'SUM',

'MAX': 'MAX',

'MIN': 'MIN',

'COUNT': 'COUNT',

'AS': 'AS',

"DESC": "DESC",

"ASC": "ASC",

}

# TOKENS

tokens = list(reserved.values()) + [

'NAME', 'COMMA', 'LP', 'RP', 'EQUALS',

'LT', 'GT', 'LE', 'GE', 'NE', 'STAR',

'NUMBER'

]

SQLParser.py中的Gramar:

Grammar

Rule 0 S' -> query

Rule 1 query -> select

Rule 2 query -> LP query RP

Rule 3 select -> SELECT list FROM table\_list opt\_where\_clause opt\_order\_clause

Rule 4 opt\_where\_clause -> WHERE condition

Rule 5 opt\_where\_clause -> empty

Rule 6 opt\_order\_clause -> ORDER BY NAME ASC

Rule 7 opt\_order\_clause -> ORDER BY NAME DESC

Rule 8 opt\_order\_clause -> ORDER BY NAME

Rule 9 opt\_order\_clause -> empty

Rule 10 condition -> NAME EQUALS NAME

Rule 11 condition -> NAME LT NAME

Rule 12 condition -> NAME GT NAME

Rule 13 condition -> NAME LE NAME

Rule 14 condition -> NAME GE NAME

Rule 15 condition -> NAME NE NAME

Rule 16 condition -> condition AND condition

Rule 17 condition -> condition OR condition

Rule 18 condition -> LP condition RP

Rule 19 condition -> NAME BETWEEN NAME AND NAME

Rule 20 condition -> NAME BETWEEN NUMBER AND NUMBER

Rule 21 condition -> NAME IN LP query RP

Rule 22 condition -> NAME EQUALS aggregate\_function

Rule 23 list -> list COMMA field

Rule 24 list -> field

Rule 25 list -> STAR

Rule 26 table\_list -> table\_list COMMA table

Rule 27 table\_list -> table

Rule 28 empty -> <empty>

Rule 29 aggregate\_function -> AVG LP NAME RP

Rule 30 aggregate\_function -> SUM LP NAME RP

Rule 31 aggregate\_function -> MAX LP NAME RP

Rule 32 aggregate\_function -> MIN LP NAME RP

Rule 33 aggregate\_function -> COUNT LP NAME RP

Rule 34 table -> NAME

Rule 35 table -> NAME AS NAME

Rule 36 table -> aggregate\_function

Rule 37 table -> aggregate\_function AS NAME

Rule 38 field -> NAME

Rule 39 field -> NAME AS NAME

Rule 40 field -> aggregate\_function

Rule 41 field -> aggregate\_function AS NAME

Rule 42 field -> STAR

对该项目进行的一些测试:

可以通过运行以下命令来执行这些测试：

**pytest tests/**

AST的生成:

SQL为

SELECT COUNT(column1) AS count\_col1, COUNT(column2) AS count\_col2, COUNT(column3) AS count\_col3

FROM table1 AS tab1, table2 AS tab2

WHERE column1 BETWEEN value1 AND value2

生成的AST:

+ QUERY

+ [SELECT]

+ [FIELDS]

+ [FIELD]

+ [COUNT]

+ column1

+ [AS]

+ count\_col1

+ [FIELD]

+ [COUNT]

+ column2

+ [AS]

+ count\_col2

+ [FIELD]

+ [COUNT]

+ column3

+ [AS]

+ count\_col3

+ [FROM]

+ [TABLES]

+ [TABLE]

+ table1

+ [AS]

+ tab1

+ [TABLE]

+ table2

+ [AS]

+ tab2

+ [WHERE]

+ [CONDITION]

+ [BETWEEN]

+ column1

+ value1

+ value2

SELECT id FROM student WHERE chinese BETWEEN 60 AND 80 生成的AST:

+ QUERY

+ [SELECT]

+ [FIELDS]

+ [FIELD]

+ id

+ [FROM]

+ [TABLES]

+ [TABLE]

+ student

+ [WHERE]

+ [CONDITION]

+ [BETWEEN]

+ chinese

+ 60.0

+ 80.0

实验要求的三个查询所对应的SQL语句:

**queries = [**

**"SELECT id FROM student WHERE chinese = MAX(chinese)",**

**"SELECT \* FROM student ORDER BY sum DESC",**

**"SELECT AVG(math) FROM student"**

**]**

查询结果:

Query 1: SELECT id FROM student WHERE chinese = MAX(chinese)

[[70605]

[70603]]

---------------------------------------------------------------------------------

Query 2: SELECT \* FROM student ORDER BY sum DESC

[[70605 131 143 144 418]

[70603 131 135 144 410]

[70609 127 139 142 408]

[70601 123 148 136 407]

[70606 126 135 140 401]

[70604 129 133 138 400]

[70602 116 143 140 399]

[70616 114 142 139 395]

[70607 115 139 135 389]

[70621 123 130 134 387]

[70611 116 142 129 387]

[70612 118 136 131 385]

[70608 122 124 139 385]

[70623 121 123 139 383]

[70610 126 115 139 380]

[70633 121 127 131 379]

[70625 119 129 130 378]

[70620 111 139 128 378]

[70619 124 108 144 376]

[70614 124 128 122 374]

[70613 121 123 128 372]

[70668 116 131 122 369]

[70636 114 124 122 360]

[70667 116 123 119 358]

[70624 116 122 118 356]

[70626 118 126 111 355]

[70629 112 109 130 351]

[70646 109 116 125 350]

[70649 114 117 118 349]

[70645 110 102 136 348]

[70635 114 113 120 347]

[70618 110 117 119 346]

[70643 113 113 119 345]

[70637 117 121 106 344]

[70617 112 105 126 343]

[70615 103 127 110 340]

[70634 108 110 119 337]

[70638 105 105 126 336]

[70622 115 111 106 332]

[70631 101 112 106 319]

[70642 98 104 116 318]

[70627 103 103 111 317]

[70641 96 130 89 315]

[70650 118 82 112 312]

[70648 105 102 105 312]

[70632 101 91 115 307]

[70630 112 90 104 306]

[70647 107 67 129 303]

[70639 109 68 126 303]

[70655 94 98 104 296]

[70644 107 76 104 287]

[70628 106 100 80 286]

[70651 105 95 82 282]

[70640 101 59 108 268]

[70656 85 95 85 265]

[70669 95 71 85 251]

[70654 97 76 71 244]

[70661 99 87 44 230]

[70659 83 61 71 215]

[70652 87 77 43 207]

[70657 82 53 62 197]

[70653 79 49 64 192]

[70660 99 21 67 187]

[70662 90 29 64 183]

[70663 78 45 47 170]

[70658 86 32 46 164]

[70664 75 23 34 132]

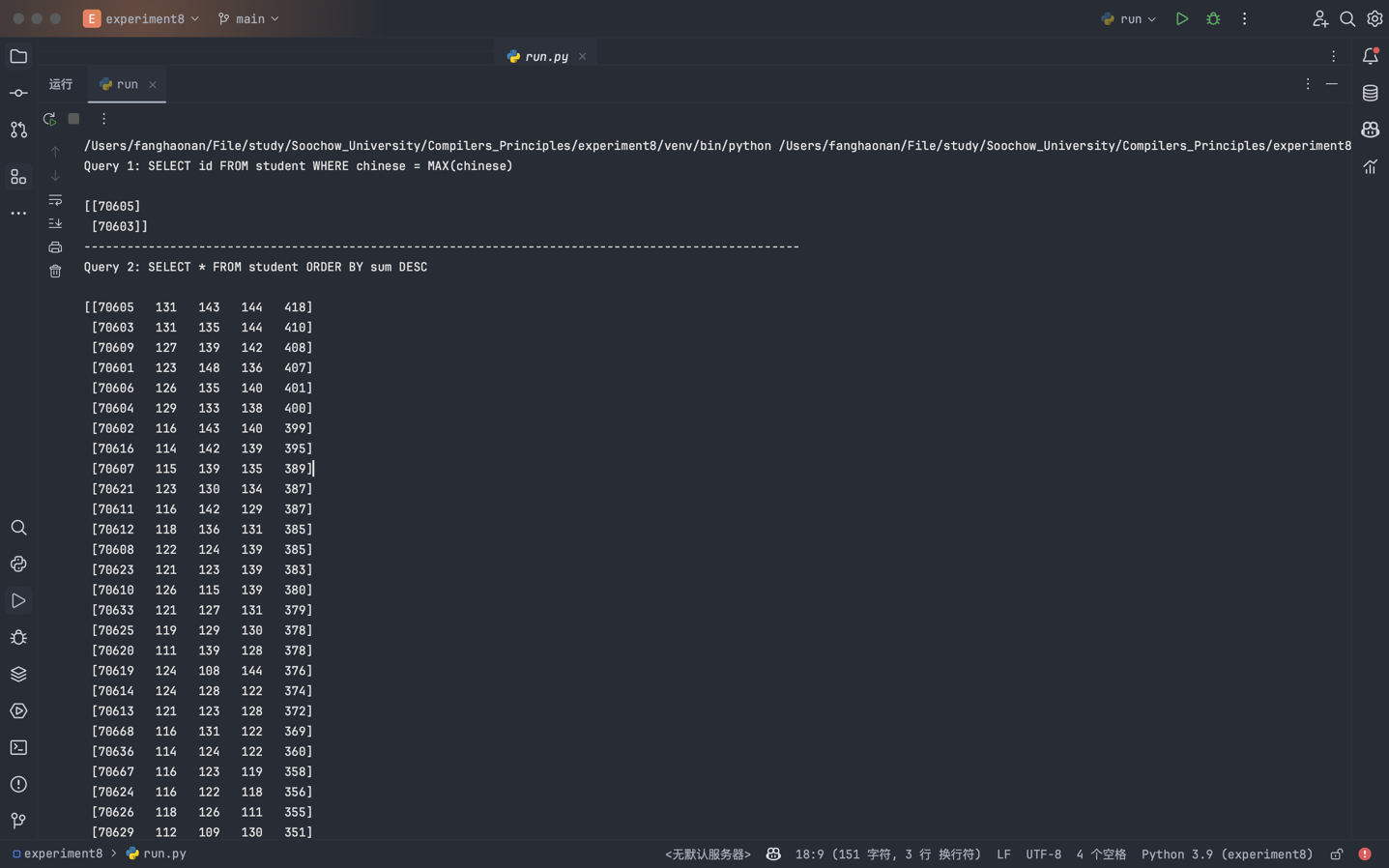
[70665 66 23 34 123]]

---------------------------------------------------------------------------------

Query 3: SELECT AVG(math) FROM student

102.82352941176471

实验结果部分截图:



1. 实验总结

技术掌握与应用：

通过本实验，我深入理解了SQL查询语言的解析和执行过程。

成功应用了ply库来扩展现有的SQL解析程序，使其能够处理更多的SQL查询语法。

学习并应用了pandas和numpy库来处理和分析CSV文件中的数据。

通过pytest编写测试用例，确保程序的正确性和可靠性。

问题解决：

在扩展SQL解析程序时，遇到了一些关于如何正确定义解析规则的问题，通过查阅ply库的文档和在线资源，成功解决了这些问题。

在处理CSV数据时，了解了如何利用pandas和numpy库的功能来简化数据处理过程，并提高数据处理的效率。

实际应用与拓展：

本实验的SQL解析程序可以作为处理和分析SQL查询语句的基础，有助于进一步开发和优化数据库查询处理系统。

通过本实验，也了解了如何通过编写测试用例来保证程序的正确性，这对于实际项目开发中保证代码质量是非常有用的。

自我反思：

本实验让我认识到了实际项目开发中遇到问题时，如何通过查阅文档和在线资源来解决问题的重要性。

实验过程中也让我认识到了编写测试用例的重要性，它不仅能帮助我们发现并修复程序中的错误，还能帮助我们理解和掌握程序的运作机制。

展望：

未来可以尝试进一步优化SQL解析程序，使其能够处理更复杂的SQL查询语句，例如支持更多的聚合函数和子查询等。

也可以探索如何将本实验中开发的SQL解析程序应用到实际的数据库系统中，以提高查询处理的效率和准确性。