**Report for Spring 2021 CS307 Project Part 1**

**Part1. Group Info and Contribution**

Name: 赵奕沣

Id: 11811525

Contribution: 50%

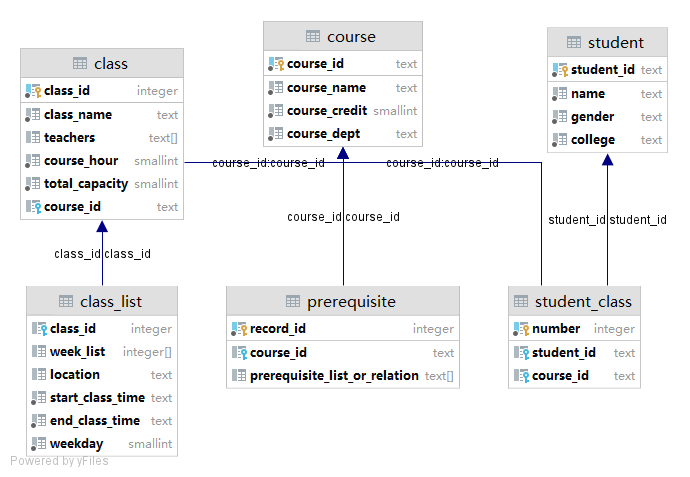
Name: 马卓远

Id: 11912412

Contribution: 50%

**Part2. Task1**

Diagram:



**Tables:**

course

|  |  |  |
| --- | --- | --- |
| course\_id | 课程编号，存储格式为text，不限长度 | 主键 |
| course\_name | 课程名称，存储格式为text，不限长度 | 非空 |
| course\_credit | 课程学分，存储格式为smallint | 非空 |
| course\_dept | 开课院系，存储格式为text，不限长度 | 非空 |

  每一个course\_id都有唯一的course\_name, course\_credit和course\_dept，因此这三个数据放在同一个表中。

class

|  |  |  |
| --- | --- | --- |
| class\_id | 班级编号，存储格式为int，值为导入顺序编号 | 主键 |
| class\_name | 班级名称，存储格式为text，不限长度 | 非空 |
| teachers | 授课老师，存储格式为text[]，可为空 |  |
| course\_hour | 课程时长，存储格式为smallint | 非空 |
| total\_Capacity | 课程容量，存储格式为smallint | 非空 |
| course\_id | 课程编号，存储格式为text，不限长度， 连接course(course\_id) | 外键 |

class\_list

|  |  |  |
| --- | --- | --- |
| class\_id | 班级编号，存储格式为int，值为导入顺序编号，  连接class(class\_id) | 外键 |
| Week\_list | 上课周，存储格式为int[]  一个班级对应多条此数据 |  |
| location | 上课地点，存储格式为text，不限长度，可为空 |  |
| start\_class\_Time | 课程开始时间，存储格式为text | 非空 |
| End\_class\_Time | 课程结束时间，存储格式为text | 非空 |
| weekday | 上课时间，存储格式为smallint，值在0-8之间 | 非空 |

prerequisite

|  |  |  |
| --- | --- | --- |
| record\_id | 记录编号，存储格式为int，值为顺序编号 | 主键 |
| course\_id | 课程编号，存储格式为 text，不限长度 | 外键 |
| prerequisite\_list  \_or\_relation | 关系为或的先修课程构成的集合，存储格式为text[]，每个课程可对应多条此记录，记录间为且关系 |  |

student

|  |  |  |
| --- | --- | --- |
| student\_id | 学生编号，存储格式为 text，不限长度 | 主键 |
| name | 学生姓名，存储格式为 text，不限长度，可重名 | 非空 |
| gender | 学生性别，存储格式为 text | 非空 |
| college | 学生所属学院名称， 存储格式为 text | 非空 |

student\_class

|  |  |  |
| --- | --- | --- |
| number | 记录编号，存储格式为 int，值为顺序编号 | 主键 |
| student\_id | 学生姓名，存储格式为 text，不限长度 | 外键 |
| course\_id | 选择课程，存储格式为 text[] | 外键 |

Postgres数据库中有三种类型储存字符串：text, character(n) 和 character varying。虽然在其他的数据库系统中，character(n)可能具有一定的性能优势，但在postgres中它们之间没有性能差别。除去当填充空白类型时增加存储空间以及检查字符串长度时，由于额外的储存成本，通常character(n)甚至会慢于另外两个。所以，为了方便起见，在设计本次数据库时，我们统一所有的字符串储存类型为text。

(https://blog.csdn.net/u013992330/article/details/76653361?utm\_source=blogxgwz9)

**Part3. Task2**

1. 导入course\_info.json

运行脚本文件 courseInfo\_import.py，通过json包将json读入为字典列表，连接数据库，再每次用循环获取合适的数据后，立刻用execute方法执行插入语句，最后提交修改。

运行时间：小于1s。

1. 导入select\_course.csv
2. Execute方法

参照导入course\_info.json文件时的思路，通过csv包将csv文件读入为列表格式后，逐条执行execute语句。由于运行时间过长，我们放弃对完整的400万行数据运行该脚本。考虑到程序运行时间和数据量之间接近线性关系，所以可以尝试用数个小样例的运行时间估算结果。在这里我们选用1万、5万、10万、25万和60万行的样例测试运行时间，并将结果进行线性拟合以得到对完整数据运行时长的估计值。

在同一台计算机上运行csv\_reader.py文件，对五个测试样本导入数据库的执行时间结果记录在表格中。使用Python skylearn中的 LinearRegression方法得到线性拟合方程为：，得分为0.9999860225557089。

T表示程序最终运行时间，单位是秒（s）

R表示数据数量，单位是千行（103 rows）

取r为3999.920，得到估计的运行时间T’ = 2800.721284s，约为47分钟。

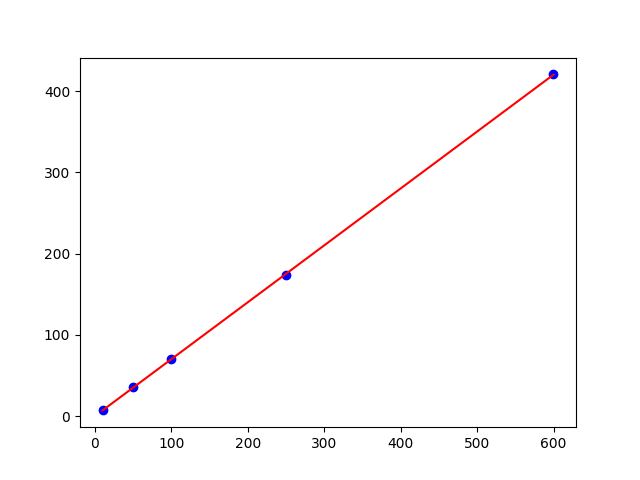


Figure 1. Linear Fit of execute

1. Executemany方法

在网上搜索资料时，有不少结显表示executemany方法执行效率是execute方法的数倍甚至数十倍。但在实际运行脚本时，我们发现两者的差异并不明显，相较于execute方法, executemany只有略微的优势，甚至有时候会慢于execute。通过测试在运行execute脚本时的样例来估算executemany的运行时长，最终发现executemany并没有相比execute在执行效率上占据显著优势。但从代码简洁性和可读性上来看，显然executemany更好，同时使用executemany也能避免在写循环执行execute语句时出现不必要的操作，从而增加额外耗时。

运行executemany.py文件，对得到的数据拟合，得到的线性拟合方程为：

，得分为0.9999040839470492。运行完整数据的估计值T’=2888.239492s，约为48分钟。

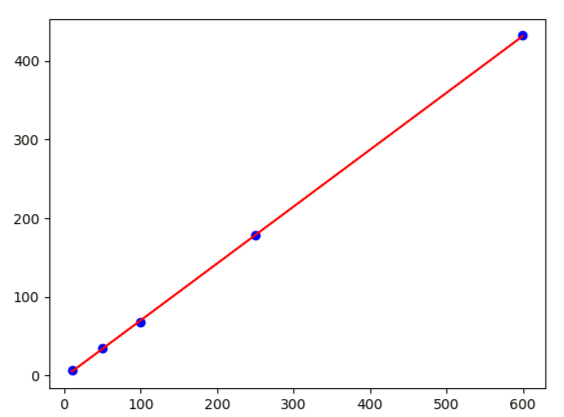


Figure 2. Linear Fit of executemany

1. Copy\_from方法

Python中的copy\_from方法利用IO技术，提高了数据传输和处理能力。用该方法插入select\_course.csv中的所有数据平均用时只有9分15秒，比前两个方法快了5倍左右。此外注意避免在循环过程中使用字符串拼接，否则会极大地降低运行效率。

1. 比较和小结

得到execute，executemany的估计运行时间值和copy\_from方法的实际运行时间值。令execute方法的运行速率为1，计算出另外两种方法的运行速率（见Figure 3）。

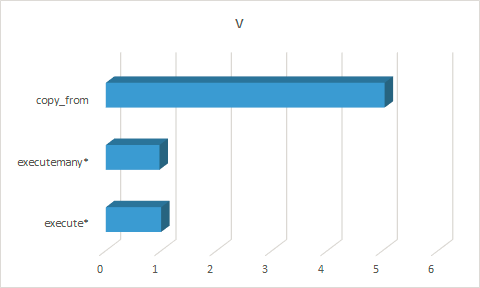


Figure 3. 三种方法执行速率比较图

Executemany方法在测试中并没有比execute方法快，可以推测两者在逻辑上并没有太大不同。而网上许多得出executemany比execute更快的结论，实际上的代码逻辑是在循环中每次执行execute语句后就执行commit语句，会极大地运行时长，所以其结论并不可靠。而利用IO技术的copy\_from方法提升速率明显，是其他方法的5倍左右。所以在实际操作中，如果追求运行速率则推荐使用copy\_from；如果追求代码简洁性，可以使用executemany。

备注：①所有脚本的运行时间的数据储存在time\_data.xlsx文件

②线性回归的数据由脚本文件LinearFit.py得到

③脚本文件csv\_reader.py使用execute方法；executemany.py使用executemany方法；copy\_from.py使用copy\_from方法

④五个测试样例分别是test1w.csv，test5w.csv，test10w.csv，test25w.csv，test60w.csv，数据来源于select\_course.csv

**Part4. Task3**

**（一）介绍基本结构和信息**

1.直接使用project提供的.json和.csv文件

2.操作系统为windows10

3.编程语言为python3

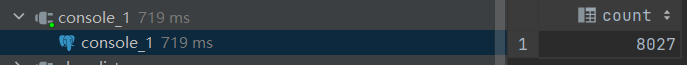
4.数据结构为list

**（二）设计比较算法和实验**

此项任务，我们选择Python中的list数据结构作为存储数据的容器，与数据库的操作效率进行对比，包括查找数据、排序数据。

**1.查找数据**

（1）进行查找学生名字效率对比，计算名字中含特定字符（刘，李，赵，邹）的学生个数















|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 包含“刘” | 包含“赵” | 包含“李” | 包含“邹” |
| 数据库执行时间（ms） | 719 | 582 | 645 | 614 |
| Python执行时间（ms） | 527 | 658 | 528 | 524 |

综合表格中四组数据，Python的执行效率与数据库相近。

（2）进行查找学生编码效率对比，计算编码中含特定数字（9，5，3，7）的学生个数











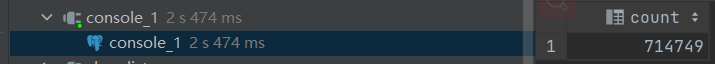




|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 包含9 | 包含5 | 包含3 | 包含7 |
| 数据库执行时间（ms） | 590 | 613 | 609 | 575 |
| Python执行时间（ms） | 744 | 705 | 772 | 656 |

综合表格中四组数据，Python的执行效率与数据库相近。

（3）进行查找学生选课信息效率对比，计算选课中含特定院系课程（PHY，MA，CS，CH）的学生个数

















|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 包含9 | 包含5 | 包含3 | 包含7 |
| 数据库执行时间（ms） | 2474 | 3842 | 3458 | 3018 |
| Python执行时间（ms） | 2154 | 2124 | 5410 | 2026 |

综合表格中四组数据，Python的执行效率与数据库相近。

**2.排序数据**

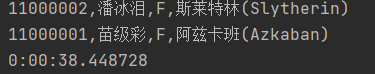
（1）根据学生选课个数进行降序排序





（2）根据学生编码进行降序排序





（3）结论：根据以上两点测试，数据库在排序运行效率相比Python有较大优势。

**3.创建角色并赋予权限**

数据库允许创建不同角色并赋予不同权限

1. student角色，有三个用户，student角色有查看course表格的权限，用户有查看存储个自选课信息表格的权限。

（个自选课信息表格为新创立的表格）

1. teacher角色，有两个用户，teacher角色有查看和改变所有表格的权限。

结论：数据库可以赋予不同用户权限，保证信息的安全性。