CS307 Project1

实验 1 班 13 组 林慧燕 邬一帆

目录

第一部分：小组信息和团队贡献

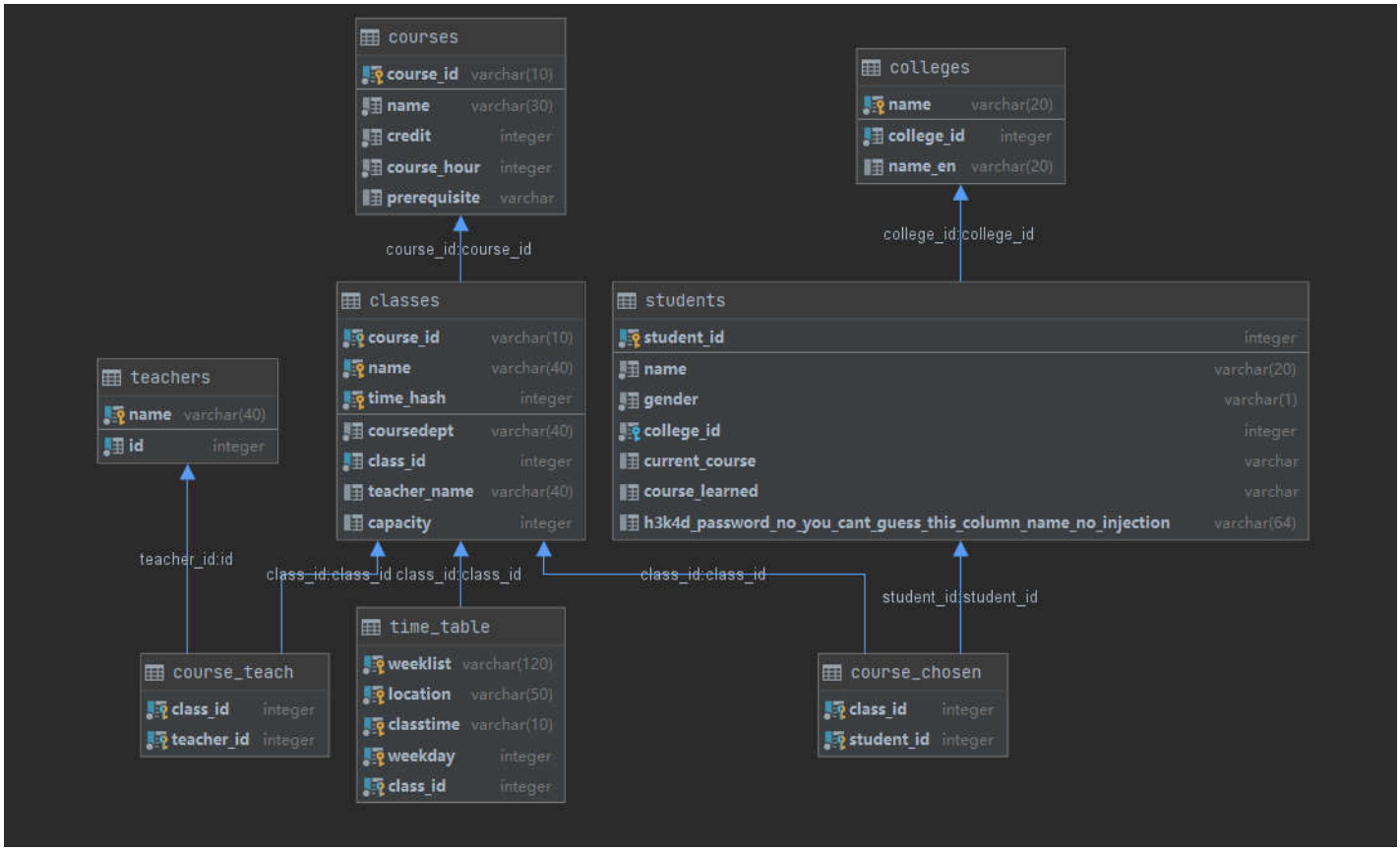
1）分工详情表

注：分工占比：50% 50%

具体工作内容	
林慧燕 50%	邬一帆 50%
设计数据库	
提高插入数据效率	处理数据、插入数据
设计测试标程	设计查询系统前后端
测试高并发和索引对效率的影响	对比硬件配置、操作系统和数据库系统对查询语句执行效率的影响
对比文件系统和数据库系统	
查找资料、分析结果、报告撰写	

第二部分：数据库设计

1）表结构图



2）先修关系处理

把先修课保留成字符串的原因

每学期的先修课都会变化，且没有固定模板。在不进行逻辑分析的前提下**保证准确，最快地处理问题**。

给定的现实数据 course.name 不唯一，甚至有大量不存在映射关系的课程名，难以合理转化课程名。

直接转化成 SQL 语句可以**最高效率**解决这个问题并**保证获得准确的结果。**

先修课需求以 sql 语句的形式存储，在查询时将学生 id 和先修需求拼接。

```
prerequisite
<null>
(course_learned like '% 理论力学I %' OR course_learned like '% 理论力学I %') AND (course
(course_learned like '% 高等数学（下）A %' OR course_learned like '% 高等数学（下） %' OR
(course_learned like '% 材料科学基础 %' OR course_learned like '% 材料科学基础 %')
<null>
<null>
(course_learned like '% 数字电路 %' OR course_learned like '% 数字电路 %' OR course_lear
```

拼接逻辑的 PHP 语句

```php
if (isset($sta[0])) {
    $sql = 'select count(*) from students where student_id= $1 and ' . $sta[0];
    $res = pg_query_params($db, $sql, array($stu_id));
```

拼接后的 SQL 查询语句

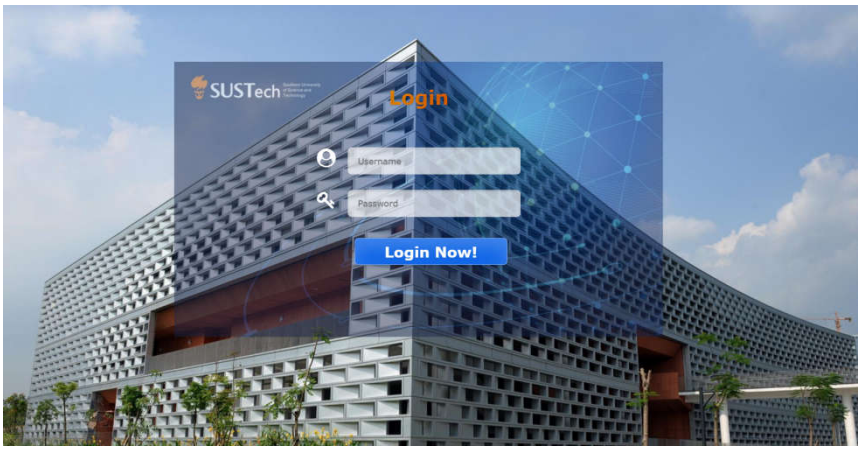
```
select count(*) from students where student_id=119129090
   AND (course_learned like '% 材料科学基础 %' OR course_learned like '% 材料科学基础 %');
```

过滤先修课不足

先修课不足，选课失败。

3）查询系统设计

①系统设计



0. 登陆界面(右图) ---------->>>>

1. 过滤已选课程

```php
$sql = 'select count(*) from(select course_id from(select class_id from course_chosen
        where student_id=$1)cid inner join classes as cl on cid.class_id=cl.class_id)oid
        inner join classes as c on c.course_id = oid.course_id where c.class_id = $2;';
$res = pg_query_params($db, $sql, array($stu_id, $cla_id));
$sta = pg_fetch_row($res);
if ($sta[0] == 0) {//!=即为过滤选课，==过滤退课
    echo "已经退选了这门课，请勿重复退课。";
```

2. 过滤已满课程

```
$sql = 'select count(*) from course_chosen where class_id=$1';
$res_0 = pg_query_params($db, $sql, array($cla_id));
$sql = 'select capacity from classes where class_id=$1';
$res_1 = pg_query_params($db, $sql, array($cla_id));
if ($sta_0 = pg_fetch_row($res_0) && $sta_1 = pg_fetch_row($res_1)) {
    if (isset($sta_0[0]) && $sta_0[0] >= $sta_1[0]) {
        echo "课程班级人数已满，选课失败。";
```

3. 过滤时间冲突课程并提示冲突课程名

```
$sql = "select count(*),y.class_id from
    (select * from time_table where class_id=$1)z
    join( select * from
    (select class_id s from course_chosen where student_id=$2)a
    join time_table b on a.s=b.class_id) y
    on z.weekday=y.weekday and (z.classtime like \"left\"(y.classtime,2)||'%'
    or z.classtime like '%'||\"right\"(y.classtime,2)) and (z.weeklist like
    y.weeklist||'%' or z.weeklist like '%'||y.weeklist or y.weeklist like
    z.weeklist||'%' or y.weeklist like '%'||z.weeklist or length(z.weeklist)>80
    or length(y.weeklist)>80 ) group by y.class_id";
$res = pg_query_params($db, $sql, array($cla_id, $stu_id));
$sta = pg_fetch_row($res);
if (isset($sta[0])) {
    $sql = "select c.name,a.name from (select course_id,name from classes
            where class_id=$1)a join courses c on a.course_id=c.course_id;";
    $res = pg_query_params($db, $sql, array($sta[1]));
    $sta_ = pg_fetch_row($res);
    echo "课程时间冲突，冲突课程：" . $sta_[0] . "-" . $sta_[1] . "，选课失败。";
```

4. 过滤先修不足课程（参见第一部分）

5. 多条件联合模糊查询及退选课

查询逻辑

```
$sql="select a.course_id,b.name,class_id,coursedept from (
(select * from courses where course_id like $1 and name like $2)a inner join
(select * from classes where teacher_name like $3 and coursedept like $4 )b
on a.course_id=b.course_id)";
$ret = pg_query_params($db, $sql, array(
    $_POST['course_id']==='Course ID'||$_POST['course_id']==""?'%':"%".strtoupper(trim($_POST['course_id']))."%",
    $_POST['course_name']==='Course Name'||$_POST['course_name']==""?'%':"%".strtoupper(trim($_POST['course_name']))."%",
    $_POST['course_teacher']==='Course Teacher'||$_POST['course_teacher']==""?'%':"%".strtoupper(trim($_POST['course_teacher']))."%",
    $_POST['course_dept']==='Course Dept'||$_POST['course_dept']==""?'%':"%".strtoupper(trim($_POST['course_dept']))."%"));
if ($ret) {
    while ($row = pg_fetch_row($ret)) {
        printTable($row,  mode: 1);
```

6. 按周次查询课程表

第1周 课程表

节次\星期	星期一	星期二	星期三	星期四	星期五	星期六	星期日
第1-2节							
第3-4节	计算机程序设计基础A 英文班-实验4班 荔园2栋101教室 胡春风			半导体材料与器件 英文班 一教501 程鑫	财务会计 英文班 荔园2栋205教室 赖舒芳		
第5-6节	机械工程导论 中英双语班 荔园6栋403 融亦鸣		宏观经济学 中英双语班 荔园2栋205教室 孙便霞		宏观经济学 中英双语班 荔园2栋205教室 孙便霞		
第7-8节			西方音乐史 中文班 一教403 艺术中心外聘老师				
第9-10节				艺术与科学大讲堂 中文班 毕宝仪			
第11-12节							

第8周 课程表

节次\星期	星期一	星期二	星期三	星期四	星期五	星期六	星期日
第1-2节							
第3-4节	计算机程序设计基础A 英文班-实验4班 荔园2栋101教室 胡春风			计算机程序设计基础A 英文班-实验4班 荔园6栋402机房 胡春风	财务会计 英文班 荔园2栋205教室 赖舒芳		
第5-6节	半导体材料与器件 英文班 一教501 程鑫		财务会计 英文班 荔园2栋205教室 赖舒芳		宏观经济学 中英双语班 荔园2栋205教室 孙便霞		
第7-8节			西方音乐史 中文班 一教403 艺术中心外聘老师				
第9-10节				艺术与科学大讲堂 中文班 毕宝仪			
第11-12节							

7. 密码修改、登出（参见下一张图片）
8. 教师显示所任课程和所带班级学生，劝退所带学生

您教授的课程

Course ID	Course Name	Teacher Name	Time and Location	Credit and Length	Prerequisite	Class Dept
MA201a	常微分方程A 中英双语习题2班	王勇	1-15周, 星期五第9-10节 一教305教室 1-15周, 星期三第1-2节 一教506教室 1-15单周, 星期一第5-6节 一教506教室	4学分, 80学时	(数学分析精讲 OR 数学分析III OR 数学分析III OR 数学分析精讲 OR 数学分析精讲) AND 线性代数II	数学系
MA201a	常微分方程A 中英双语班习题1班	王勇	1-15周, 星期四第9-10节 一教305教室 1-15周, 星期三第1-2节 一教506教室 1-15单周, 星期一第5-6节 一教506教室	4学分, 80学时	(数学分析精讲 OR 数学分析III OR 数学分析III OR 数学分析精讲 OR 数学分析精讲) AND 线性代数II	数学系

您课程中的学生

Course Name	Student Name	Student Gender	Student id	Student College	Options
常微分方程A中英双语习题2班	测试人员-	M	119129090	阿兹卡班	劝退

9. 管理员执行 sql 语句（非最高权限），修改任意用户密码

name	college_id	name_en
赫奇帕奇	3	(Hufflepuff)
阿兹卡班	1	(Azkaban)
格兰芬多	5	(Gryffindor)
斯莱特林	2	(Slytherin)
拉文克劳	4	(Ravenclaw)

②权限管理

第三部分：数据导入

1）脚本编写

使用 python 的 json 和 re 库完成脚本生成。

读取 json 文件到列表

```
def load():
    data=[]
    with open('course_info.json', 'r', encoding='utf-8') as f:
        data = json.load(f)
    return data;
```

数据处理（以 course 表的数据为例，处理先修课）

```
def change(x): #处理前的数据 (生物化学 I（生物大分子） 或者 生物化学 I（生物大分子）) 并且 (细胞生物学 或者 细胞生物学) 并且 普通生物学实验
    r=""
    x=x.replace("或者","OR");
    x=x.replace("并且","AND");
    l=re.split(r'([\(\) ])',x) #通过 （ ） 和 空格 分割，保留分隔符
    for i in l:
        if i!='OR' and i != 'AND' and i != ')' and i != '(' and i != ' ' and i != '':
            r+=("course_learned like ''% {} %''".format(i)) #保存为 sql 语句
        else:
            r+=(i)
    return r  #返回的数据 (course_learned like '% 生物化学 I（生物大分子） %' OR course_learned like '% 生物化学 I（生物大分子） %') AND (course_learned like '% 细胞生物学 %' OR course_learned like '% 细胞生物学 %') AND course_learned like '% 普通生物学实验 %'
```

```
def insert_course(data): #生成 sql 语句
    course=[]
    for i in data:
        course.append('''INSERT INTO public.courses (name, course_id, credit,
         course_hour, prerequisite) VALUES ('{}','{}',{},{},{});'''
        .format(i['courseName'],i['courseId'],i['courseCredit'],
        i['courseHour'],'null' if i['prerequisite']==None else "'"+
        str(change(str(i['prerequisite'])))+"'"))
    course=set(course)
    return course
```

写 sql 文件

```
with open('ins.sql', 'w+', encoding='utf-8') as f:
    f.writelines(dat)
    f.close()
```

2）提高导入效率

检测导入效率，我们只采用了学生信息，包含五个信息，姓名、性别、学院、学号、已学课程，近 400 万条数据我们分别截取了其中的 10 万条和 100 万条作为测试样例。根据老师提供的 DEMO，我们可以很容易的得出五种导入数据的办法，并且提供了 **unlogged** 加速和**索引**加速。

1.对已有方法的比较分析

①Awful

用动态 sql 语句插入数据，每次插入都要打开关闭一次数据库。因为速度过慢，因此只导入了 100 条数据，但能够看出这种导入方式的低效。

```
C:\Users\86158\.jdks\corretto-1.8.0_275\
100 records successfully loaded
Loading speed : 12 records/s

Process finished with exit code 0
```

②Very bad

准备插入语句前打开数据库，等所有数据插入完后再关闭，大大减少数据库连接次数，提高了极大的效率。

```
C:\Users\86158\.jdks\corretto-1.8.0_2
100000 records successfully loaded
Loading speed : 2283 records/s

Process finished with exit code 0
```

③Bad

采用预编译语句，将 sql 语句提前加载，运行时只需要传递参数，加快效率。

```
C:\Users\86158\.jdks\corretto-1.8.0_275
100000 records successfully loaded
Loading speed : 3677 records/s

Process finished with exit code 0
```

④Average

开启事务将所有 sql 加载完成后，一次性写入磁盘。

```
C:\Users\86158\.jdks\corretto-1.8.0_275\bin\java.exe ...
100000 records successfully loaded
Loading speed : 11453 records/s

Process finished with exit code 0
```

⑤Good

设置 batch_size，使用批处理，同时将多条语句发送给数据库 server，减小时间消耗。

```
C:\Users\86158\.jdks\corretto-1.8.0_275\bin\java.
100000 records successfully loaded
Loading speed : 59171 records/s

Process finished with exit code 0
```

2. 探索其它加速方法

⑥unlogged 加速

重新创建表格，关闭日志

```
Create unlogged table students(
    name            varchar(30) not null,
    gender          varchar(1)  not null,
    college         varchar(30) not null,
    student_id      integer     not null,
    current_course varchar(64) not null  );
```

继续使用⑤中的 loader 代码

```
C:\Users\86158\.jdks\corretto-1.8.0_275\bin\j
1000000 records successfully loaded
Loading speed : 65798 records/s

Process finished with exit code 0
```

关闭日志后提速约 **10.17%**。

⑦索引加速

CREATE INDEX test1_id_index ON students (student_id);

```
[2021-04-12 00:19:23] Connected
postgres> CREATE INDEX test1_id_index ON students (student_id)
[2021-04-12 00:19:29] completed in 5 s 287 ms
```

在高并发（详见第四部分）的情况下使用 count 查询代码测试

n = 100；//同时访问查询一百次数据库

加索引前

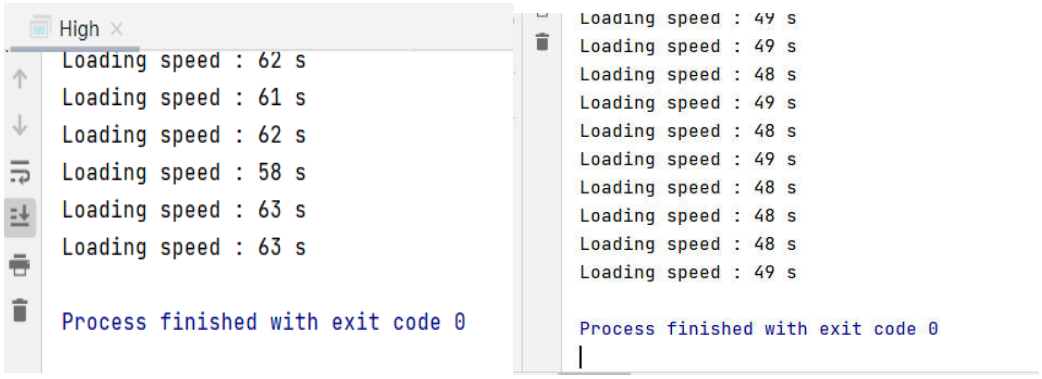
```
Loading speed : 14 s
Loading speed : 13 s
Loading speed : 14 s
Loading speed : 13 s
Loading speed : 13 s
Loading speed : 13 s
Loading speed : 13 s
Loading speed : 13 s
Loading speed : 13 s
Loading speed : 13 s
Loading speed : 13 s
Loading speed : 13 s
Loading speed : 13 s

Process finished with exit code 0
```

加索引后

```
Loading speed : 11 s
Loading speed : 11 s
Loading speed : 11 s
Loading speed : 11 s
Loading speed : 11 s
Loading speed : 11 s
Loading speed : 11 s
Loading speed : 12 s
Loading speed : 13 s
Loading speed : 12 s
Loading speed : 12 s
Loading speed : 12 s
Loading speed : 13 s
Loading speed : 12 s
Loading speed : 12 s
```

n = 1000；

加索引前　　加索引后



加索引后提速约 **16.67%**。

第四部分：测试对比

1）环境设计

①外部环境设计（操作系统、硬件架构、数据库种类）

7 种不同配置下的 13 个数据库（配置详情请参照附表 1）：

- Debian & Windows 两大系列 4 种操作系统
- CISC & RISC 两大系列 3 种硬件架构
- 同系统下不同系统版本横向对比

配置编号	1	2	3	4	5	6	7
操作系统	ubuntu18.04	ubuntu 5.4.0-1032-raspi	win7	win10	ubuntu12.04	kali-686-pae	win10
CPU	1物理1i逻辑E5	1物理4i逻辑ARM-aarch64	1物理4i逻辑i79650	1物理4i逻辑i79650	1物理4i逻辑i79650	1物理4i逻辑i79650	8物理16i逻辑AMD7-4800
内存	2GB	4GB	4GB	4GB	4GB	4GB	16GB
磁盘	SCSI	SD	SCSI	SCSI	SCSI	SCSI	SATA
mysql版本	mysql 5.7.33	mysql 8.0.23	mysql 5.6.32	mysql 8.0.22	mysql 5.5.32	mysql 5.5.44	
pgsql版本	psql 10.16	psql 12.6	psql 10.16 x86	psql 10.16 x64	psql 9.1.24	psql 9.4.3	psql 12.1

mysql 数据格式与 pgsql 接近，只需要将已生成的 sql 文件去掉注释，指定 varcahr 长度即可导入，保证数据一致。采用已有数据的前 100 000 条进行测试。

②内部环境设计（文件系统、数据库系统及并发）

用 python 脚本多线程模拟多次请求，O(n)复杂度对读入内存中的文本遍历查询，记录处理全部请求所用时间

```
def run(i,dat):
    time_start=time.time()
    cal(11111111+i,dat)
    time_end=time.time()
    print('totally cost',time_end-time_start)


if __name__ == "__main__":
    print("loading..")
```

```
    a=sp(load())
    print("start..")
    threads=[]
    for i in range(1,300):
        try:
            t=threading.Thread(target=run,args=(i,a))
            threads.append(t)
            t.start()
        except:
            print ("线程异常")
    for t in threads:
        t.join()
```

- 高并发处理

使用 3.1.2.6 中的表格，向数据库同时发起 n 次计数查询请求，记录处理完全部数据的时间

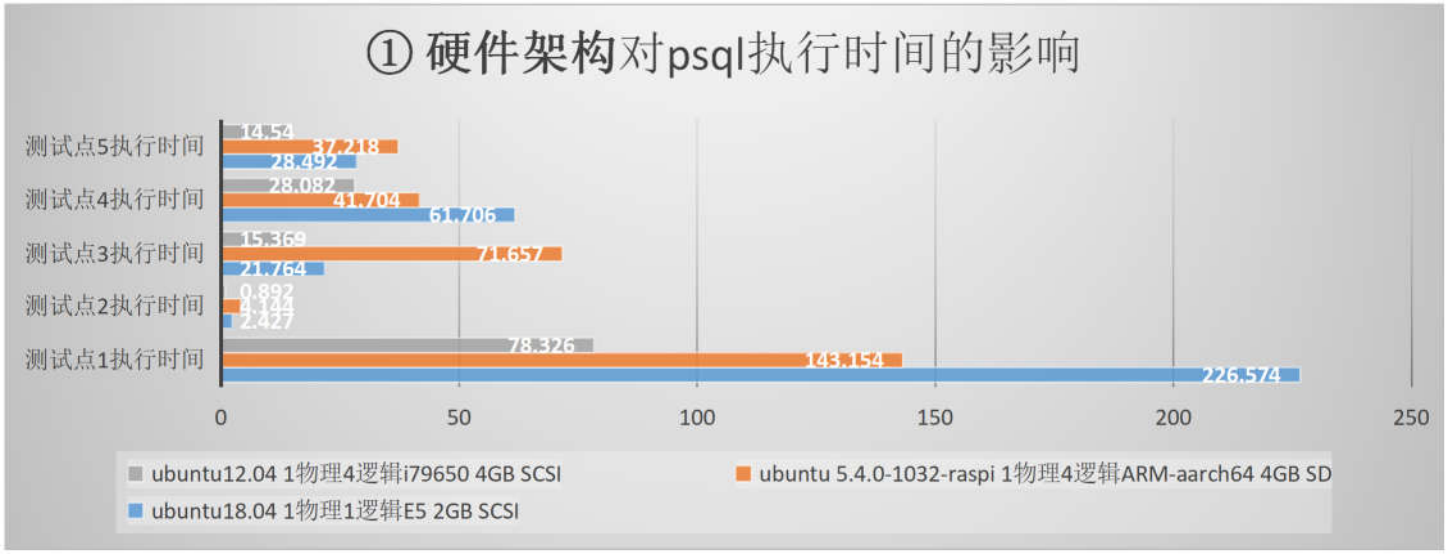
```
import java.util.Properties;
import java.sql.*;
public class High {
    public static void main(String[] args) {
        for (int i = 1; i <= 100; i++) {//同时发出 100 次申请
            new ThreadLoader(11111111 + i).start();
        }
    }
    static class ThreadLoader extends Thread {
        public int id;
        public ThreadLoader(int id) {
            this.id = id;
        }
        public void run() {
            String url = "jdbc:postgresql://" + "localhost" + "/" + "postgres";
            Properties props = new Properties();
            props.setProperty("user", "postgres");
            props.setProperty("password", "lihongji520");
            try {
                long start;
                long end;
                Connection con = DriverManager.getConnection(url, props);
                con.setAutoCommit( false );
                start = System.currentTimeMillis();
                Statement stmt=con.createStatement();
                stmt.executeQuery("select count(*) from students where student_id > " + id);
                con.commit();
                end = System.currentTimeMillis();
                System.out.println("Loading speed : " +  (end - start)/1000 + " s");
                stmt.close();
                con.close();
            } catch (SQLException se) {
```

```
                System.err.println("SQL error: " + se.getMessage());
            }
        }
    }
}
```

2）测试分析

①外部环境对查询执行效率影响的分析

测试方法： 在命令行下执行测试脚本，mysql 关闭查询缓存，使用 show profiles;查询耗时，psql 使用 EXPLAIN ANALYZE 分析耗时。



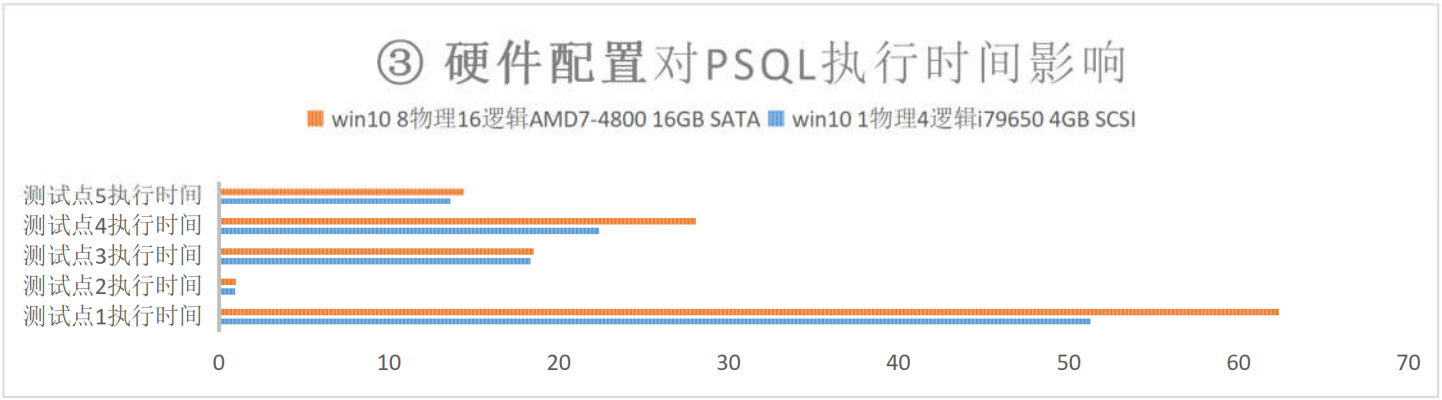
psql 的守护进程(Postmaster)会 fork 多个辅助进程，这可能是产生差异的重要原因。

对于内存更大内核更多的橙色，速度较单处理器单核的蓝色(cortex-a7)有一定提升，而对于与蓝色同配置的灰色(I7 的每个核心拥有独立 MMU)处理多进程效率比橙色又有提升。

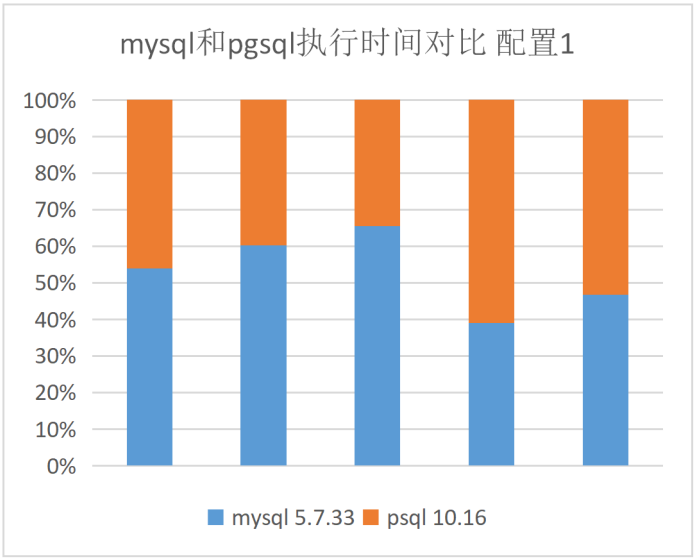
综上，提升处理器性能和内存对提升 psql 执行效率有一定影响(但并非绝对)，而提升处理器并行能力可以显著提升 psql 的执行效率。

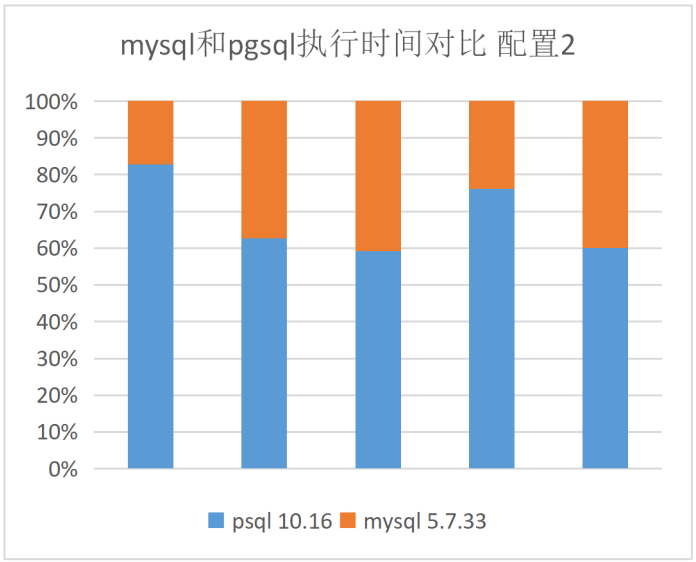
蓝色 32 位操作系统耗时显著数倍于橙灰黄 64 位系统

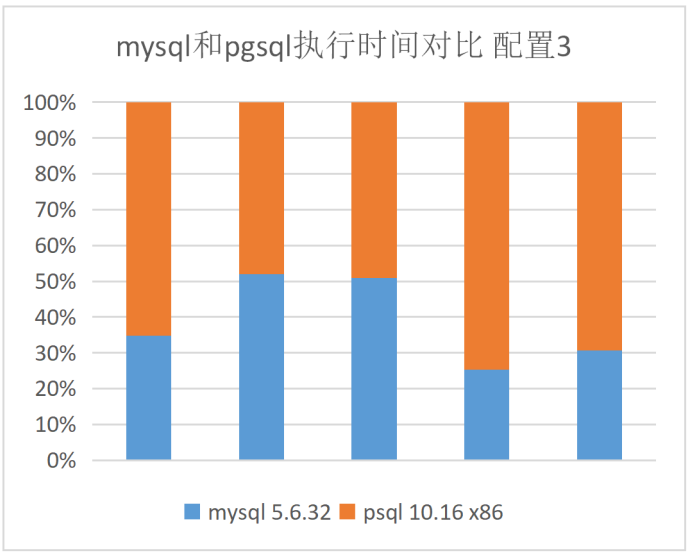
在同为 64 位系统的情况下，多个系统耗时并无明显区分，在计算密集的场景下 windows 操作系统略快于其它系统。

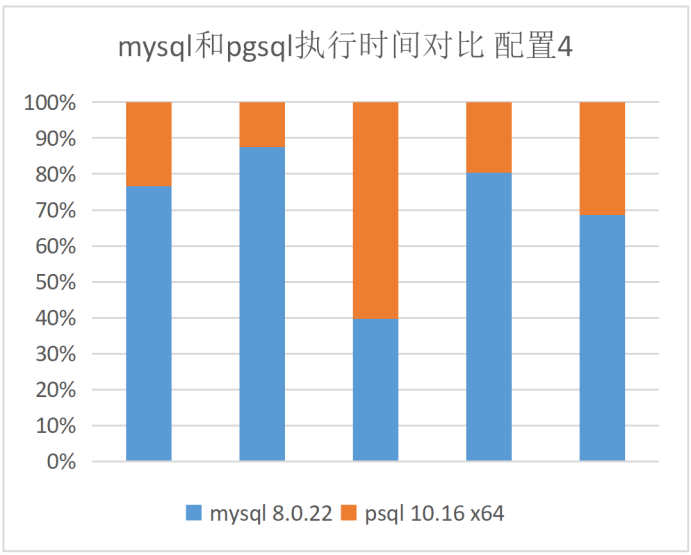


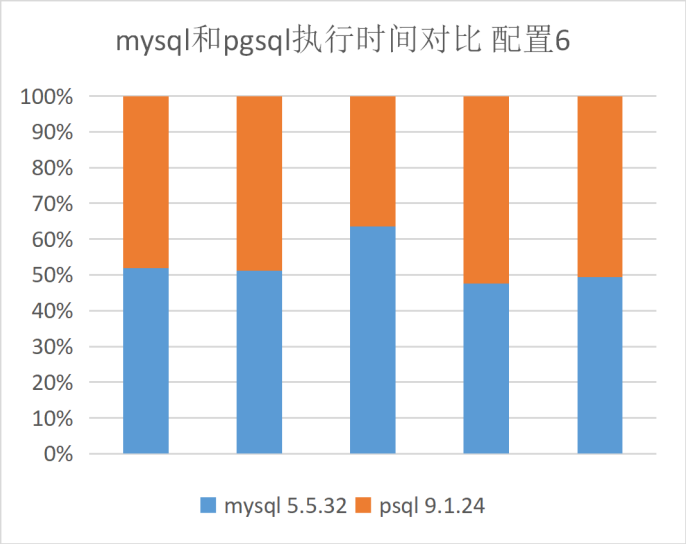
在硬件架构对比中我们认为提升内存和 CPU 性能并不一定提升 psql 的执行简单查询语句效率，在这个对比中再次得出相似的结果。

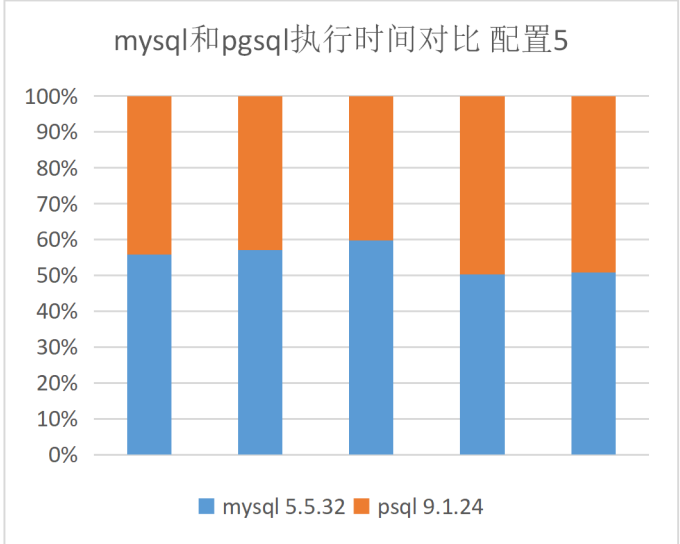








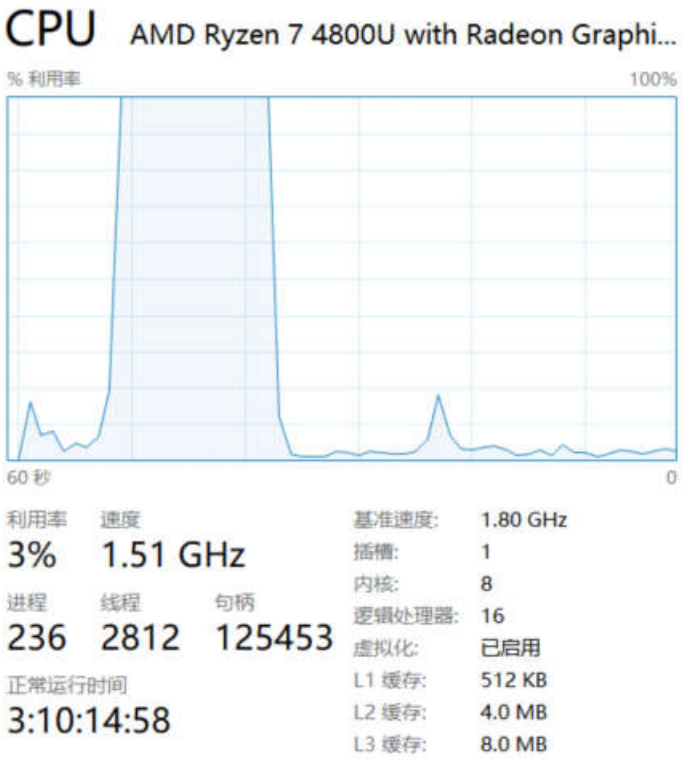


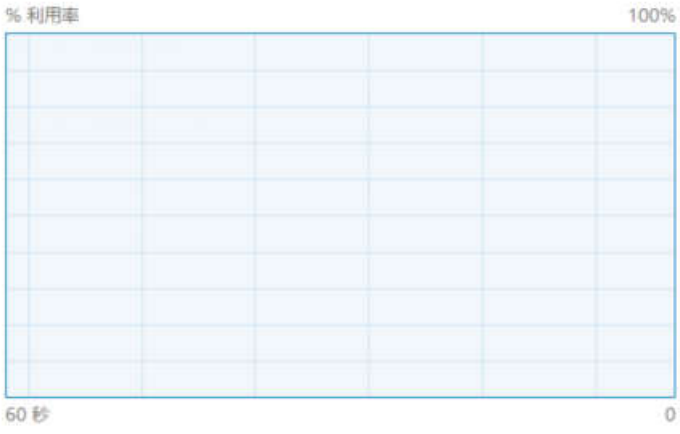


上图中占比越多表示相对执行时间越长，即超过 50%代表比另一种数据库执行耗时长。除了在配置 3 的低配 x86 下 mysql 快过了 psql，其他场景都是 psql 略优于甚至明显优于 mysql。

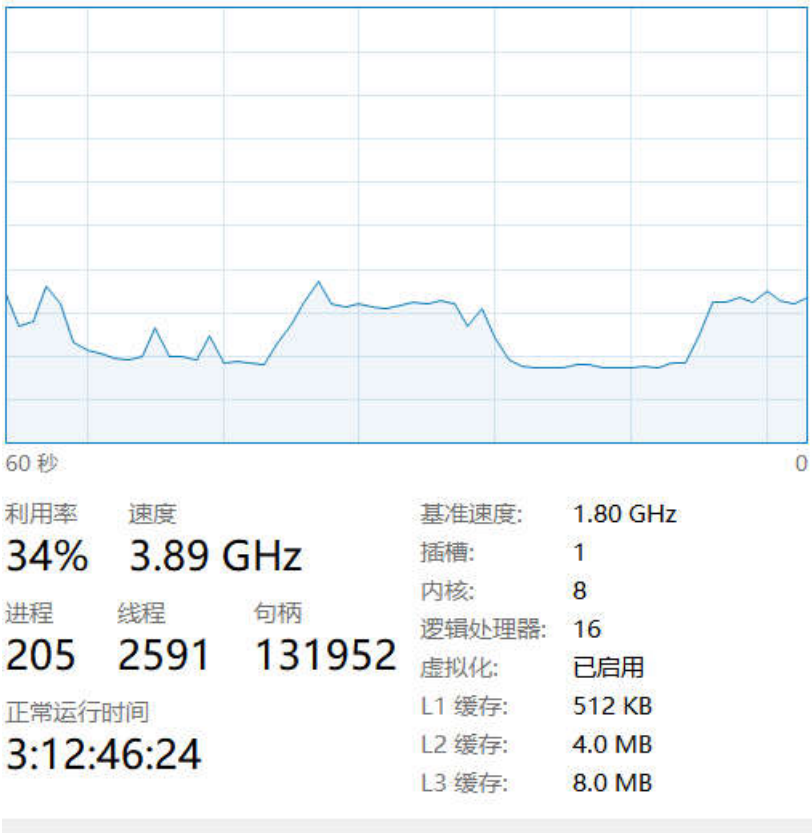
②内部环境对查询执行效率影响的分析

数据库处理高并发时 CPU 的占用情况

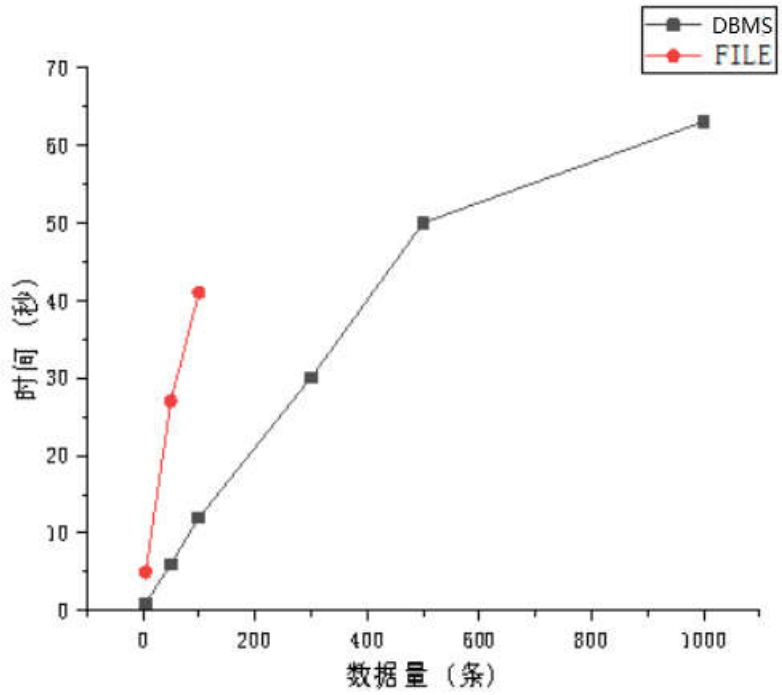




文件系统处理高并发时 CPU 的占用情况



数据库处理高并发处理请求耗时和文件搜索处理高并发请求耗时的拟合曲线



可以发现，数据库系统处理高并发请求时间远远小于文件搜索的处理时间。在请求次数到达一定数量后，数据库的耗时**逐渐平滑**，增长缓慢。而文件搜索耗时始终接近**正比**。在数据量达到 100 条时，文件系统的耗时已经是数据库的 4 倍；**随着数据量增大，数据库的耗时远小于文件搜索的耗时。**

造成差异的原因可能有①数据库对系统资源调用高效，而普通的文件搜索没有充分使用系统资源②数据库内部使用 B+Tree 等高效数据结构，优化了处理效率。

总结

本项目设计了一个包含课程数据，教师学生信息的关系数据表，处理了诸如先修关系，时间冲突的请求并设计了合理的权限管理系统，实现了一个类似教务系统的前端，可以退选课，修改密码，打印课程表，联合条件模糊查询等功能，并具有教师端和管理端。

随后本项目通过脚本处理并导入了数据，探索了 7 种可以有效提升数据导入效率的方法如事务管理，日志和索引。

最后本项目对比测试了高并发环境下的数据库和文件系统，不同硬件系统，不同硬件架构，不同系统环境和不同品牌数据库环境下的查询执行效率，并得出以下结论：

- psql 的性能发挥有依赖处理器并行计算能力的特点，且 64 位操作系统执行效率明显高于 32 位。而查询语句本身执行不需要消耗大量内存资源，所以提高内存配置可能不会有效提升 psql 执行效率。mysql 在低配情况下对普通查询处理效率略高于 psql，对于其它一般场景，psql 性能表现普遍优于 mysql。
- 使用 DBMS 进行数据管理比使用文件储存有非常大的优势，对高并发请求的处理也更为高效。

第五部分：附录

项目 git 仓库(暂时私有)：github.com/GhostFrankWu/SUSTech_CS307_DB_Projects

项目在线 demo：https://suste.cn/

1）附加文件说明

main 文件夹：

create.sql 主数据库的建表语句
ins_others.sql 除学生外其它数据的插入语句
ins_student.sql 学生数据的插入语句
script.py 处理原始数据生成 sql 语句的 python 脚本（非直接生成）

sc_difference 文件夹：

保存了 task3 中所有测试结果的截图，图片中的数据已录入附表

SQL-psql.sql 初始化部署 psql 数据库的 sql 脚本
SQL-mysql.sql 初始化部署 mysql 数据库的 sql 脚本
SQL-psql_time.txt 测试 psql 执行时间的代码
SQL-mysql_time.txt 测试 mysql 执行时间的代码

web 文件夹：

查询系统的源代码，所有 php 文件为独立完成，整个项目没有使用任何框架。

misc 文件夹:

其它部分*非必要*的 Project 文件

2）附表

附表① 完整测试数据							
配置编号	1	2	3	4	5	6	7
操作系统	ubuntu18.04	ubuntu 5.4.0-1032-raspi	win7	win10	ubuntu12.04	kali-686-pae	win10
CPU	1 物理 1 逻辑 E5	1 物理 4 逻辑 ARM-aarch64	1 物理 4 逻辑 i79650				8 物理 16 逻辑 AMD7-4800
内存	2GB	4GB	4GB				16GB
磁盘	SCSI	SD	SCSI				SATA
数据库版本	mysql 5.7.33	mysql 8.0.23	mysql 5.6.32	mysql 8.0.22	mysql 5.5.32	mysql 5.5.44	\
测试点 1 执行时间	265.028	690.19075	191.65525	167.208	98.683	82.567	\
测试点 2 执行时间	3.67825	6.969	3.69275	6.5265	1.18475	0.87725	\
测试点 3 执行时间	41.2775	103.649	27.83375	12.031	22.73425	23.287	\
测试点 4 执行时间	39.45875	133.06425	32.18525	91.66725	28.445	23.76375	\
测试点 5 执行时间	24.9905	55.816	17.91825	29.705	15.0285	13.796	\
数据库版本	psql 10.16	psql 12.6	psql 10.16 x86	psql 10.16 x64	psql 9.1.24	psql 9.4.3	psql 12.1
测试点 1 执行时间	226.574	143.154	359.028	51.278	78.326	76.587	62.337
测试点 2 执行时间	2.427	4.144	3.423	0.932	0.892	0.838	0.97
测试点 3 执行时间	21.764	71.657	26.76	18.342	15.369	13.327	18.502
测试点 4 执行时间	61.706	41.704	94.911	22.375	28.082	26.133	28.061
测试点 5 执行时间	28.492	37.218	40.688	13.599	14.54	14.176	14.405
测试点 1 准备时间	0.314	0.346	0.342	2.773	N/A	0.197	0.231
测试点 2 准备时间	0.687	3.715	0.673	0.551	N/A	0.399	0.671
测试点 3 准备时间	0.1	0.667	0.072	0.083	N/A	0.064	0.207
测试点 4 准备时间	0.168	0.363	0.156	0.179	N/A	1.407	0.142
测试点 5 准备时间	0.074	0.209	0.056	0.08	N/A	0.043	0.107
注：							
①时间单位均为毫秒(ms)　　②MySQL 与 PostgreSQL 都是高可配置的，这里我们使用他们的默认参数 ③配置 3-6 实际由 win10-i79650, 24G 内存 SATA 盘 4 物理 16 逻辑 CPU 的实体机通过 VMware15.5pro 执行测试							