OOP大作业：数据库

（1） 第二阶段开始之前

每组应该收到4份其他组的代码。组的编号已被打乱，请大家自觉维护不要讨论打分相关的问题。

每组应该首先阅读其他4个组的代码，联系第二阶段需求，分析代码是否易于进一步开发。你需要对这些代码进行排序，并简要分析每份代码的优劣性。排名不得并列，对每组代码的评论至少50字。你的排序及评价会由助教评估是否合理，占你大作业总得分的5%。别人对你的评价占你大作业总得分的10%。

你继续开发使用的代码应当是你排序中的第一位。你也可以选择直接使用自己的代码，这时第二阶段得分会受到**分数x0.8**的惩罚。但相应地，你可以节省看代码的时间，通过完成更多的功能提高自己的得分。

（2）第二阶段需求

在以下的开发过程中，你需要尽量保持第一阶段其他组的接口不被修改（可以增加）。在第二阶段开发结束后，第一阶段的测试SQL脚本应当不需改动，直接能够正确运行。（因为往往在真实项目中，修改接口会牵一发而动全身，容易浪费大量时间。）如果实在需要改动，会按照改动幅度和合理性酌情扣分。最后你需要提交原来的main.cpp和完成第二阶段的main.cpp，我们会再次测试第一阶段的数据。

基础需求（占25%，如果只完成基础需求，大作业总分上限为85%）：

1. 你需要提交第一阶段的main.cpp，并保证仍能通过第一阶段的测试，**占5%**。注意你拿到的代码可能仍有bug，你应该尝试修复你使用的这份代码中可能存在的bug。
2. 实现数据的导入和导出语句，**占5%**：
    1. 数据导出，`SELECT ... INTO OUTFILE`可以把被选择的行写入一个文件中。输出不能是一个已存在的文件，以防止文件数据被篡改；输出的文件中每行为一组数据，没有表头，不同列之间用`\t`分割；样例见`db_save.sql`和`outputu_file`。

        ```
        SELECT * INTO OUTFILE 'output_file' FROM oop_info;
        ```

    2. 数据导入，`LOAD DATA LOCAL INFILE 'dump.txt' INTO TABLE mytbl`，以上实例中将从当前目录中读取文件 dump.txt ，将该文件中的数据插入到当前数据库的 mytbl 表中。样例见`db_load.sql`和`db_load.out`。

        ```
        LOAD DATA INFILE 'dump.txt' INTO TABLE mytbl(b, c, a);
        ```

        支持输入文件中列的顺序与表原始顺序不同，输入文件均使用`\t`作为分割符号；这里我们省略了`LOCAL`关键词；
3. 实现`COUNT(expression)`函数，返回查询的记录总数，expression 参数是一个字段或者 * 号，**占5%**；样例见`count.sql`和`count.out`。为了简化起见，`COUNT`函数只需要支持在SELECT语句中使用，不需要支持`whereClauses`子句。

```
SELECT COUNT(*) from oop_info;
```

`COUNT`可用于统计表中行的总数，也可以用于统计表中某一列不为空的个数（即过滤NULL值）`COUNT(stu_name)`。

4. 实现分组语句，**占5%**：
    1. `GROUP BY` 语句根据一个或多个列对结果集进行分组。在分组的列上我们可以使用 COUNT 等函数。

```
SELECT stu_name, COUNT(*) from oop_info GROUP BY stu_name;
```

上面的语句完成了同名学生的统计，并整理输出。

5. 实现排序语句，**占5%**：
    1. 如果我们需要对读取的数据进行排序，我们就可以使用 MySQL 的 `ORDER BY` 子句来设定你想按哪个字段哪种方式来进行排序，再返回搜索结果。样例见`group_order.sql`和`group_order.out`。

```
SELECT stu_name, COUNT(*) from oop_info GROUP BY stu_name ORDER BY
COUNT(*);
```

将姓名从次数少到次数多依次输出。由于本阶段有些时候输出的表格无主键，这时候输出的顺序会存在不确定的情况，因此都需要加上`ORDER BY`进行控制，如果这时候值依旧出现相同，那么任意顺序都算正确，我们会使用spj进行判断。

拓展需求（拓展需求会按照完成情况、接口设计进行打分）：

1. 实现支持多表的`whereClauses`子句，多表之间使用`,`分割，实现多表占**5%**：

```
SELECT oop_info.name, oop_score.score
FROM oop_info, oop_score
WHERE oop_info.stu_id = oop_score.stu_id
AND oop_info.join_year = oop_score.exam_year;
```

2. 实现`UNION`操作符，UNION 操作符用于连接两个以上的 SELECT 语句的结果组合到一个结果集合中。多个 SELECT 语句会删除重复的数据。

```
SELECT stu_name FROM oop
UNION
SELECT stu_name FROM fop
ORDER BY stu_name;
```

使用`UNION ALL`则不会删除重复的数据，该功能**占5%**。

3. 实现更多的SQL数字函数并将其在示例代码中**有机**地用起来，参考 http://www.runoob.com/mysql/mysql-functions.html ，一个函数2%，最多加**10%**（只需要能在SELECT语句中使用即可）；使得数字函数能够`whereclauses`子句当中使用，再加**5%**（保证已实现的数字函数能够使用即可，包括`COUNT`函数）。
    1. 例如：`MIN(expression)`，`MAX(expression)`等。
4. 扩展支持的数据类型，实现日期和时间类型：`DATE`，`TIME`，每个时间类型有一个有效值范围和一个"零"值，当指定不合法的MySQL不能表示的值时使用"零"值，以及实现两个日期函数（`ADDDATE(d,n)`、`ADDTIME(t,n)`），**占10%**。新的数据类型需要支持已有的比较操作符。
5. 实现SQL算术运算符，即加、减、乘、除和取余，占**5%**。在除法运算和模运算中，如果除数为0，将是非法除数，返回结果为NULL。
6. 实现SQL逻辑运算符，即逻辑非（NOT）、逻辑与（AND）、逻辑或（OR）和逻辑异或（XOR），占**5%**。这里要注意NULL在所有的逻辑运算下都返回NULL。
7. 支持多表的连接`JOIN`，**占10%**，包括：
    1. `INNER JOIN`（内连接,或等值连接）：获取两个表中字段匹配关系的记录。
    2. `LEFT JOIN`（左连接）：获取左表所有记录，即使右表没有对应匹配的记录。
    3. `RIGHT JOIN`（右连接）： 与 `LEFT JOIN` 相反，用于获取右表所有记录，即使左表没有对应匹配的记录。
8. 实现数据库的存档功能（退出后启动仍可以看到原先的数据），加**5%**，如果在此基础上可以做到每一次操作都修改对应数据文件，再加**5%**，如果通过使用特定算法使得修改的复杂度低于线性复杂度，最多再加**10%**。
9. 支持`LIKE`子句进行模糊搜索，使用百分号 % 字符来表示任意字，最多再加**10%**。
10. 支持数据库的远程访问，服务器能够连接客户端机器，接受客户端发来的指令，最多**加15%**。
11. 其他你觉得有意义的改进。

    注意：对于多数拓展需求，你需要自己编写测试程序，保证展示你的结果。

（3） 项目限制

只能使用C++完成，禁止使用第三方库（不包括STL或编译器自带的库）。

（4） 提交要求

第12周周日(5.19)：你对其他组的排序及评价

第17周周日(6.23)：

1. 提交你编写的数据库，请遵守OOP的设计规范。
2. 保证代码能够运行第一阶段的测试脚本，第二阶段的基础需求，以及提供你完成的其他拓展功能的测试程序。
3. 给出一份实验报告，展示你们最终完成的功能，测试程序运行的结果。并分析实现的好和需要改进的地方。
4. 在readme.md或readme.txt中写明程序的运行环境。（你的库在跨平台上最好能在跨平台下运行，并且依赖项尽量少，否则我们将很难运行你的代码。）
5. 给出一个说明文档（markdown或word），写清数据库的结构、封装、接口。（字数不限，目标是能让别人在最短时间内明白项目结构，太短太长都不太好。）