

实验课 08

实验8-1 SQL语言初步

SQL是Structureed Query Language（结构化查询语言）的缩写，它是一种专门用于与数据库进行沟通的语言。

与其他编程语言（例如C，Java，Python）不同，SQL只有很少的关键词，目的是提供一种从数据库中读写数据的简单有效的方法。

目前大多数企业用户选择免费的MySQL作为自己的数据库，我们来简单学习一点适用于MySQL数据库的SQL语句，当然绝大部分语句也可应用于其他数据库管理系统。

假设我们有一张表，表名为 `school_grade`，这张表存储着学生的学号、班级以及修读每一门课程的成绩。

id	stuNo	class	courseNo	grade
1	5011	1	220	97
2	5011	1	221	91
3	5011	1	222	89
4	5012	1	220	85
5	5013	2	221	72
6	5012	1	221	93
7	5014	2	220	52
8	5013	2	222	64

步骤1 选取表的字段（列）

`SELECT` 语句用于选取表的某列或某几列。为了使用 `SELECT` 语句，必须给出两条信息：选什么字段、从哪里选取。

我们现在从示例表中选取学生学号、课程代码以及成绩这3列信息。

```
SELECT stuNo,courseNo,grade FROM school_grade;
```

若执行多条SQL语句，每条语句后必须以分号结束；多数数据库管理系统不要求在单条SQL语句后添加分号，但是在每条SQL语句中添加分号是一个很好的习惯。

SQL语句不区分大小写，所以上面的SQL语句中 `SELECT` 是和 `select` 是相同的，不过对SQL语言的关键字大写，表名和字段名小写是一个很好的习惯。

SQL语句可以写成长长的一行，也可以分写在多行，例如

```
SELECT stuNo,courseNo,grade
FROM school_grade;
```

执行该条SQL语句后，会显示表 `school_grade` 全部行的学生学号、课程代码和成绩。

stuNo	courseNo	grade
5011	220	97
5011	221	91
5011	222	89
5012	220	85
5013	221	72
5012	221	93
5014	220	52
5013	222	64

输出结果的列名默认为数据表的列名，你也可以自定义输出的列名。

```
SELECT stuNo AS 学号, courseNo AS 课程号, grade AS 成绩 FROM school_grade;
```

输出结果如下

学号	课程号	成绩
5011	220	97
5011	221	91
5011	222	89
5012	220	85
5013	221	72
5012	221	93
5014	220	52
5013	222	64

如果想选取所有列，可直接用 * 选取，例如

```
SELECT * FROM school_grade;
```

步骤2 检索不同的值

现在我们来看一下有哪些班级，写出选取班级名字段的SQL语句：

```
SELECT class FROM school_grade;
```

class
1
1
1
1
2
1
2
2

事实上一个班中有很多同学，我们只想简单查看不重复的班级，这时候我们需要用到 `DISTINCT` 关键字，指示数据库只返回不同的值。

```
SELECT DISTINCT class FROM school_grade;
```

class
1
2

步骤3 限制结果

现实中一张数据表可能包含成千上万条记录，如果返回全部记录可能造成系统缓慢，我们可以利用 `LIMIT` 关键字限制输出的行数。

例如我们想输出全部字段，从第2条记录开始的3条记录：

```
SELECT * FROM school_grade LIMIT 1,3;
```

注意：第一个被检索的行为第0行，所以第2条记录被称为第1行。

id	stuNo	class	courseNo	grade
2	5011	1	221	91
3	5011	1	222	89
4	5012	1	220	85

步骤4 使用注释

SQL中一般使用两个减号作为行内注释的开始，例如

```
SELECT * FROM school_grade; --这是一条注释
```

多行注释同C/C++语言，例如

```
/* 第一行注释
第二行注释 */
SELECT * FROM school_grade;
```

步骤5 排序检索数据

现在我想输出全部字段，**并按照成绩排序**。

这时候就需要用到 `ORDER BY` 子句，后接字段名及排序方式，例如

```
SELECT * FROM school_grade ORDER BY grade;
```

id	stuNo	class	courseNo	grade
7	5014	2	220	52
8	5013	2	222	64
5	5013	2	221	72
4	5012	1	220	85
3	5011	1	222	89
2	5011	1	221	91
6	5012	1	221	93
1	5011	1	220	97

不过默认排序方式为升序排序，如果想降序排序，还需要后接 `DESC` 关键字

```
SELECT * FROM school_grade ORDER BY grade DESC;
```

id	stuNo	class	courseNo	grade
1	5011	1	220	97
6	5012	1	221	93
2	5011	1	221	91
3	5011	1	222	89
4	5012	1	220	85
5	5013	2	221	72
8	5013	2	222	64
7	5014	2	220	52

你也可以对多个字段进行排序，例如当成绩相同时再按班级升序排序，我们可以执行这条SQL语句：

```
SELECT * FROM school_grade ORDER BY grade DESC, class;
```

步骤6 过滤数据

数据库表中一般包含大量的数据，很少检索表中的所有行，大多数情况下我们只关心满足某些条件的记录，这时候我们需要用到 `WHERE` 子句筛选出满足设定条件的记录。

例如我们现在想查看学号为5011同学的所有课程成绩，也就是说限定条件为 `stuNo='5011'`，我们写出以下SQL语句：

```
SELECT * FROM school_grade WHERE stuNo='5011';
```

id	stuNo	class	courseNo	grade
1	5011	1	220	97
2	5011	1	221	91
3	5011	1	222	89

我们也可以选出有成绩不合格的学生学号，课程代码和成绩：

```
SELECT stuNo, courseNo, grade FROM school_grade WHERE grade < 60;
```

也可以选出成绩介于70至80之间的所有记录：

```
SELECT * FROM school_grade WHERE grade BETWEEN 70 AND 80;
```

还可以添加逻辑，例如选取1班中成绩不合格的记录：

```
SELECT * FROM school_grade WHERE class = 1 AND grade < 60;
```

步骤7 汇总数据

我们经常需要汇总数据而不用把它们实际检索出来，为此SQL提供了一些专门的函数，可以用于

- 确定记录行数(COUNT)
- 确定某些行的最大值(MAX)/最小值(MIN)/总和(SUM)/平均值(AVG)

我们用一些实际例子看一下。

例如我们现在想查看课程代码为220的平均成绩

```
SELECT AVG(grade) FROM school_grade WHERE courseNo = '220';
```

AVG(grade)
78

查看1班学生的数量（注意一个学生有多条记录，需要计数不同内容的行的数量）

```
SELECT COUNT(DISTINCT stuNo) FROM school_grade WHERE class = 1;
```

步骤8 分组数据

现在我们想查看每门课程的平均成绩，在上一步中我们学会了如何汇总数据，现在我们学习如何对各子集的数据进行汇总。

`GROUP BY` 用于将数据分为多个逻辑组，对每个组分别进行聚集计算。**注意：**`GROUP BY` 子句必须出现在 `WHERE` 子句之后，`ORDER BY` 子句之前！

```
SELECT courseNo,AVG(grade) FROM school_grade GROUP BY courseNo;
```

courseNo	AVG(grade)
220	78
221	85.33333333333333
222	76.5

我们还可以对分组后的数据进行过滤，例如我们在刚才的基础上增加一条限制条件：显示平均分大于80的课程成绩平均分。这时候就需要利用 `HAVING` 子句了。

```
SELECT courseNo,AVG(grade) FROM school_grade GROUP BY courseNo HAVING AVG(grade) > 80;
```

courseNo	AVG(grade)
221	85.33333333333333

你可能会想既然是限制条件为什么不使用 `WHERE` 子句呢？例如

```
/*错误示范*/
SELECT courseNo,AVG(grade) FROM school_grade WHERE AVG(grade) > 80 GROUP BY courseNo;
```

首先，`WHERE` 过滤的是行而不是分组，`HAVING` 过滤的是分组；其次，`WHERE` 在数据分组前进行过滤，而 `HAVING` 是在数据分组后进行过滤，也就是说 `WHERE` 先排除掉不符合过滤条件的行，通过聚集函数计算后再使用 `HAVING` 子句排除不符合过滤条件的分组。

步骤9 小结

我们回顾一下 `SELECT` 语句中子句的顺序：

- `SELECT`
- `FROM`
- `WHERE`

- GROUP BY
- HAVING
- ORDER BY

我们对数据库的四大基本操作包括[增加](#)、[删除](#)、[修改](#)和查询，其中查询是我们最经常进行的操作，其他三类操作可点击前面的链接自己学习。

实验8-2 在Python中执行SQL语句

SQL语句一般在数据库管理系统的用户客户端执行，当然我们日常浏览网页时填写表单的过程中后台也在执行相应的SQL语句。

我们可以在Python中通过执行SQL语句获取数据，并对其继续加工处理。

PyMySQL是在Python 3.x版本中用于连接MySQL服务器的一个库，通过它可以方便地获取数据库中的数据。

步骤1 安装PyMySQL库

PyMySQL是第三方提供的库，我们需要在conda中安装。

启动Anaconda PowerShell Prompt，切换至你的环境（如有），键入 `conda install pymysql` 后回车安装。

步骤2 连接数据库

对数据表进行操作之前，我们需要先连接数据库。

一般一台主机中会安装有数据库管理系统（DBMS），例如主机 `cdb-r2g8f1nu.bj.tencentcdb.com` 中安装有MySQL 5.7服务器端，开放 `10209` 端口供远程连接。

MySQL服务器中会包含多个数据库，每个数据库会包含多张数据表，同时系统会设置账户控制用户的访问和操作权限。

例如现在我们连接这台服务器，服务器中有一个数据库名为 `dase_intro_2020`，内含实验8-1用到的 `school_grade` 表，并且拥有一个账户名为 `dase2020`，密码 `dase2020` 的用户，该用户有 `dase_intro_2020` 的查询权限。现在我们尝试连接该数据库。

```
import pymysql

db = pymysql.connect(host = "cdb-r2g8f1nu.bj.tencentcdb.com", port = 10209, user
= "dase2020", password = "dase2020", database = "dase_intro_2020")
cursor = db.cursor() # 使用 cursor() 方法创建一个游标对象 cursor, 执行SQL语句都是通过游
标对象实现

sql = "SELECT VERSION();" # 该SQL语句返回MySQL的安装版本，用以确定是否成功连接服务器

cursor.execute(sql) # 执行SQL语句
result = cursor.fetchone() # 获取单条数据
print(result)
```

```
('5.7.18-20170830-log',)
```

若出现版本信息，则说明数据库连接成功，可进行查询等后续操作。

步骤3 数据库查询示例

Python查询Mysql使用 `fetchone()` 方法获取单条数据, 使用 `fetchall()` 方法获取多条数据。

我们以实验8-1的步骤1的查询案例示范这两种方法的使用。

```
sql = "SELECT stuNo,courseNo,grade FROM school_grade;"
cursor.execute(sql)
result = cursor.fetchone()
print(result)
```

```
('5011', '220', 97.0)
```

若SQL语句查询结果有多行, 每调用一次 `fetchone()` 会返回查询结果的一行。我们再调用一次查看一下。

```
result = cursor.fetchone()
print(result)
```

```
('5011', '221', 91.0)
```

若想获取所有查询结果, 请使用 `fetchall()` 方法。

```
sql = "SELECT stuNo,courseNo,grade FROM school_grade;"
cursor.execute(sql)
result = cursor.fetchall()
print(result)
```

```
((('5011', '220', 97.0), ('5011', '221', 91.0), ('5011', '222', 89.0), ('5012', '220', 85.0), ('5013', '221', 72.0), ('5012', '221', 93.0), ('5014', '220', 52.0), ('5013', '222', 64.0))
```

可以看到使用 `fetchall()` 方法后会将所有查询结果放入一个元组中, 这个大元组的元素是每一行的查询值。

若对数据库进行增加、删除或修改操作, 还需要执行 `commit()` 方法, 具体使用方法可在[这里](#)查看。

实验练习08

数据库 `dase_intro_2020` 有一张名为 `bicycle_train` 的表, 该表存储了两个城市在不同因素影响下某小时共享单车租用的数量。该表所有值均为整型数据, 表结构为:

- id 记录编号, 无其他意义
- city 城市代号, 0为北京, 1为上海
- hour 小时, 代表时间
- is_workday 是否为工作日, 0为否, 1为是
- temp_air 大气温度, 单位为摄氏度
- temp_body 体感温度, 单位为摄氏度
- weather 天气代号, 1为晴天, 2为多云或阴天, 3为雨天或雪天
- wind 风级, 数值越大代表风速越大
- y 该小时内共享单车被租用的数量

请编写Python程序，连接到数据库（主机地址、用户名及密码见实验8-2），查询以下问题，并合理处理数据，以适当的方式将结果输出至屏幕。

1. 从第18条记录开始的5条记录；
2. 数据表中风级(wind)取值范围是多少；
3. 满足城市为北京市，10时，晴天，无风或1级风，租用单车数量不小于100条件下大气温度的平均值；

参考答案：20.6

4. 满足城市为北京市，10时，晴天，无风或1级风，租用单车数量不小于100条件下大气温度的方差；

提示：需要在Python中对返回的数据进行后续处理

参考答案：37.64

5. 分城市显示工作日雨雪天单车租用总量，并降序排序；

参考答案：上海为9106。北京为8084

6. 分别查询17时至19时每小时上海市在工作日且体感温度不大于10摄氏度时租用单车的平均值（四舍五入至整数），并且结果按升序排序。

参考答案：17时65辆；18时63辆；19时42辆