第五章　数据管理

本章导读

世界是充满数据的世界，如何更好地组织管理应用数据，是长期技术热点。数据库（Database）是存储存放数据的仓库，可以存储大量数据，是一种常见的数据管理技术。但数据库不能随意存储数据，必须按照一定规则，否则将导致增加删除修改查询等效率低下难以有效应用。

行列表和键值对是最常见数据组织形态。行列表由多行多列组成，第一行是表头，确定了每一列可容纳的数据种类，或者说是属性列，描述某个对象的属性（property，attribute）及其取值。由一个或多个行列表构成的数据库称之为关系数据库（Relational Database）。

键值数据库是一种非关系数据库，类似前面学习过的字典dict。键值数据库是键值对集合的存储，其中键是唯一标识符。键和值都可以是简单值如整数字符串等，也可以是复杂的对象等任何内容。

长期以来，数据库就是指关系数据库，但随着数据规模和种类的增多，传统的关系数据库越来越难以胜任，逐步发展出了各种非关系数据库。关系数据库用SQL（Structured Query Language，结构化查询语言）查询数据，非关系数据库则常被称为NoSQL数据库。

流行的关系数据库主要有：SQLite、MySQL、PostgreSQL、SQLSever、Oracle、DB2等等。非关系数据库主要：MonoDB、HBASE、Redis等。本章以Python内置的SQLite为基础，其相关知识同样适用于其他关系数据库，且SQLite不仅适用于Python，也适用于其他多种语言，如Java、C/C++等。

学习目标：

1. 掌握SQL语句；
2. 掌握Python常用数据库管理方法；
3. 掌握数据库编程应用；

本章目录

第一节　快速了解

在现实生活中，规范的行列表很常见，如各种统计表格等。图5-1以简化的图书馆借书为例，共有3个表格，图书表用于登记图书馆的每一本图书，如图中上部所示，取名为tblBook；学生表用以登记学生信息，如左下图所示，取名为tblStudent；借阅关系如右下图所示，名为tblBorrow。每个表都有表头行和数据行所示，表头的每一列常被称为属性（attribute）或者字段（field），数据行称之为记录（record）。

表名：tblBook

ISBN	bookName	bookPrice	publisher	publishDate	bookAmount
978-7-5470-3213-8	鲁迅作品集：朝花夕拾	13.80	万卷出版公司	2014	6
978-7-5470-3199-5	鲁迅作品集：华盖集	15.80	万卷出版公司	2014	5
978-7-5470-3122-3	阅读·图书馆	34.80	万卷出版公司	2014	4
978-7-5059-8607-7	生命圆舞曲	29.00	中国文联出版社	2014	9
978-7-5108-2338-1	中华易经普通读本	34.00	九州出版社	2013	6
978-7-5402-3442-3	李英辉书画作品集	58.00	北京燕山出版社	2014	3
978-7-5034-4749-5	博客村官	68.00	中国文史出版社	2014	3
978-7-5402-3138-5	时间在这时候慢下来	26.00	北京燕山出版社	2013	7
978-7-5059-8060-0	当代精英诗人三百家	99.00	中国文联出版社	2013	8
978-7-5126-3204-2	仙游仙境仙梦仙艺：仙子故园情	32.00	团结出版社	2014	4

表名：tblStudent

studID	Name	Gender
20161361	李迷	男
20151383	姬逸飞	男
20170164	梁红春	女
20171231	徐建建	男
20160102	林金凤	女
20150532	何彬彬	女
20171402	王明杰	男
20160323	赵睿然	男

表名：tblBorrow

borrowID	studID	ISBN	borrowDate	returnDate
1	20150123	978-7-5502-0702-8	2015-10-12	None
2	20150123	978-7-5112-1137-8	2015-10-12	None
3	20150122	978-7-5502-0702-8	2016-03-02	2016-05-10
4	20150122	978-7-5502-2905-1	2016-03-02	None
5	20150122	978-7-5112-1137-8	2016-03-02	2016-05-10
6	20160122	978-7-5463-1279-8	2017-10-12	2017-12-01
7	20160122	978-7-5472-0889-2	2017-10-12	None
8	20160122	978-7-5472-0883-0	2017-10-12	2017-12-01

图5-1　数据表格样例

在上图的三个表格中，ISBN是图书编号，每种图书都有唯一编号，不同的ISBN就是不同的书；studID是学生编号，相当于学号，其中前4位表示学生入学年份，第5位用以区分初中生还是高中生，其中0表示初中生1表示高中生，后续3位是序号；borrowID是借书编号，每次借书的每本书都有唯一编号。在tblBorrow中，borrowDate表示借书日期，returnDate表示还书日期。

类似表格有很多，绝大部分都可以通过关系数据库予以很好解决，其基本操作有：创建(Create)、查询(Retrieve)、更新(Update)和删除(Delete)，简称CRUD，其对应的操作语言称之为SQL(Structured Query Language的简写，含义是结构化查询语言)。ISO（国际标准化组织）先后于1989年和1992年通过了SQL89和SQL92国际标准。Python基本库提供的SQLite是一种嵌入式小型数据库（虽说小，但也能高效处理TB级数据），其概念和操作同样适用于其他数据库，如MySQL、Oracle、SQL Server、DB2等。

例程5-1是在Python中使用数据库SQLite的简单示例。使用SQLite前，必须引入Python标准库如例程第3行所示。在第5行，通过sqlite3对象的connect()函数建立起与lib.db数据库文件的联系。**注意：SQLite是文件型数据库，其他数据库大多为服务型数据库。文件型数据库应用较为方便，将文件拷贝到指定位置配合相应程序即可使用。**

SQL语句通过SQLite的数据库链接对象connection执行，在例程dbConn即为SQLite的数据库链接对象，其函数execute()用于执行各种SQL语句。在SQL语句中，select使用频率最高，用于查询数据库中的数据。如例程第8行（第12行与之完全相同）所示，select和from之间表示选出的字段或者属性，from确定从哪个表选择数据。在例程中，将从tblBook中选择数据，虽然tblBook有6个属性，但此处仅选择3个或者说3列。

选出的每一条或者说每一行数据称之为记录（record），多条记录可以称之为记录集。在例程第8行，选出的记录集存储于变量dbRec之中，可以用for-in循环遍历所有成员。从图5-2可以看出，每条记录为tuple，可以用所学的Python知识访问其中的每一个数据项。另外，还可以通过其fetchall()函数将其转换为list，如例程第12行所示，也可以直接用list()将其转换为list类型。

例程5-1

```
第1行  #文件名为：SQLite00.py
第2行
第3行  import sqlite3 #使用sqlite的前提
第4行
第5行  dbConn=sqlite3.connect("lib.db") #创建SQLite3的对象
第6行  #lib.db是已经存储样例数据，是SQLite的数据库文件
```

```
第7行
第8行   dbRec=dbConn.execute("select ISBN,bookName,publisher from tblBook")
第9行   for everyRec in dbRec:
第10行      print(everyRec)
第11行
第12行  dbRec=dbConn.execute("select ISBN,bookName,publisher from tblBook")
第13行  recList=dbRec.fetchall()
第14行  print(recList)
第15行
第16行  dbConn.close() #关闭数据库连接，也就关闭了数据库
```

```
C:\PythonCode>python SQLite00.py
('978-7-5470-3213-8', '鲁迅作品集：朝花夕拾', '万卷出版公司')
('978-7-5470-3199-5', '鲁迅作品集：华盖集', '万卷出版公司')
('978-7-5470-3122-3', '阅读·图书馆', '万卷出版公司')
('978-7-5059-8607-7', '生命圆舞曲', '中国文联出版社')
('978-7-5108-2338-1', '中华易经普通读本', '九州出版社')
('978-7-5402-3442-3', '李英辉书画作品集', '北京燕山出版社')
('978-7-5034-4749-5', '博客村官', '中国文史出版社')
('978-7-5402-3138-5', '时间在这时候慢下来', '北京燕山出版社')
('978-7-5059-8060-0', '当代精英诗人三百家', '中国文联出版社')
('978-7-5126-3204-2', '仙游仙境仙梦仙艺：仙子故园情', '团结出版社')
[('978-7-5470-3213-8', '鲁迅作品集：朝花夕拾', '万卷出版公司'), ('978-7-5470
-3199-5', '鲁迅作品集：华盖集', '万卷出版公司'), ('978-7-5470-3122-3', '阅读
·图书馆', '万卷出版公司'), ('978-7-5059-8607-7', '生命圆舞曲', '中国文联出
版社'), ('978-7-5108-2338-1', '中华易经普通读本', '九州出版社'), ('978-7-540
2-3442-3', '李英辉书画作品集', '北京燕山出版社'), ('978-7-5034-4749-5', '博
客村官', '中国文史出版社'), ('978-7-5402-3138-5', '时间在这时候慢下来', '北
京燕山出版社'), ('978-7-5059-8060-0', '当代精英诗人三百家', '中国文联出版社'
), ('978-7-5126-3204-2', '仙游仙境仙梦仙艺：仙子故园情', '团结出版社')]
```

图5-2　例程5-1执行后的部分结果

例程5-2第5行被执行的SQL语句多了where子句，同时被选择的属性（字段或列）是*。*表示将选择所有属性，在例程中即tblBook中的所有属性。where相当于条件，在例程中的含义是将publisher值等于“团结出版社”的记录筛选出。*和前面的属性名称（如例程5-1第8行的ISBN、bookName和publisher）相当于列筛选。

例程5-2

```
第1行   #文件名为：SQLite00.py
第2行   import sqlite3 #使用sqlite的前提
第3行   dbConn=sqlite3.connect("lib.db") #创建SQLite3的对象
第4行
第5行   dbRec=dbConn.execute("select * from tblBook where publisher='团结出版社'")
第6行   recList=list(dbRec) #转换为list类型
第7行   print(dbRec) #输出全部list类型记录
第8行
第9行   dbRec.close() #关闭记录集
第10行  dbConn.close() #关闭数据库连接，也就关闭了数据库
```

例程5-3使用了order by子句，可以对选出的记录集按指定的一个或多个属性排序，且可指定降序（desc）或默认的升序（asc）方式排序。在例程中第5行是按ISBN升序排序，第10行是按ISBN的降序排序，如果ISBN相同，则按bookPrice升序排序。order by和where子句不是排斥关系，如例程第13行所示，则是对记录按publishDate筛选，然后根据ISBN和bookPrice排序。

例程5-3

```
第1行 #文件名为：SQLite00.py
第2行 import sqlite3 #使用sqlite的前提
第3行 dbConn=sqlite3.connect("lib.db") #创建SQLite3的对象
第4行
第5行 dbRec=dbConn.execute("select * from tblBook order by ISBN")
第6行 recList=dbRec.fetchall() #转换为list类型
第7行 print(recList)
第8行
第9行 dbRec=dbConn.execute("select * from tblBook order by ISBN desc,bookPrice asc")
第10行 recList=dbRec.fetchall() #转换为list类型
第11行 print(recList)
第12行
第13行 dbRec=dbConn.execute("select * from tblBook where publishDate='2014' order by ISBN desc,bookPrice as
c")
第14行 recList=dbRec.fetchall() #转换为list类型
第15行 print(recList)
第16行
第17行 dbRec.close() #关闭记录集
第18行 dbConn.close() #关闭数据库连接，也就关闭了数据库
```

例程5-4应用了group by子句用于分组，在例程第5行按publisher分组，统计每个组内的数据。group by一般和聚集函数一起使用，如例程中的count（计数），sum（求和）、avg（求平均），其他常用的还有max（求最大值）、min（求最小值）。

group by可以通过having自带条件，对经过聚集函数生成的记录集进行过滤筛选，如例程第9-10行所示。另外，对select的属性名称可以再命名，如代码中“publisher as 出版社”即将“publisher”属性命名“出版社”，其中的as关键字可以省略如“avg(bookPrice) 均价”所示。

例程5-4

```
第1行 #文件名为：SQLite00.py
第2行 import sqlite3 #使用sqlite的前提
第3行 dbConn=sqlite3.connect("lib.db") #创建SQLite3的对象
第4行
第5行 dbRec=dbConn.execute("select publisher,count(*),avg(bookPrice),sum(bookAmount) from tblBook group by
publisher")
第6行 recList=dbRec.fetchall() #转换为list类型
第7行 print(recList)
第8行
第9行 dbRec=dbConn.execute("""select publisher as 出版社,count(*) as 种类,avg(bookPrice) 均价
第10行 from tblBook group by publisher having 均价>20 order by 出版社""")
第11行 recList=dbRec.fetchall() #转换为list类型
第12行 print(recList)
第13行
第14行 dbRec.close() #关闭记录集
第15行 dbConn.close() #关闭数据库连接，也就关闭了数据库
```

SQL语句能实现多表查询，如例程5-5所示。在例程第5-7行代码中，select的数据来自3个表，分别是tblBook、tblBorrow和tblStudent，可以查询出每本借阅情况，即知道借书人学号、姓名、书名、书号、借阅日期、还书日期等。多表查询必然涉及到表与表之间的联系关系，可通过where或join两种模式。例程第5-7行代码是采用where模式，即在where条件中设置表与表之间的联结关系，如tblBook.ISBN=tblBorrow.ISBN即设定tblBook中的ISBN与tblBorrow中ISBN相等的记录筛选出。

采用join模式，则通过join-on设定表与表的联结关系。在例程第11-14行代码中，“from tblBook inner join tblBorrow on tblBook.ISBN=tblBorrow.ISBN”表示tblBook将与tblBorrow相联，联结的条件是tblBook.ISBN=tblBorrow.ISBN。早期SQL语句采用where模式，现代SQL倾向于join on。

例程5-5

```
第1行  #文件名为：SQLite00.py
第2行  import sqlite3 #使用sqlite的前提
第3行  dbConn=sqlite3.connect("lib.db") #创建SQLite3的对象
第4行
第5行  dbRec=dbConn.execute("""select tblBook.ISBN,bookName,tblStudent.studID,Name,borrowDate,returnDate
第6行  from tblBook,tblBorrow,tblStudent
第7行  where tblBook.ISBN=tblBorrow.ISBN and tblStudent.studID=tblBorrow.studID""")
第8行  for everyRec in dbRec:
第9行      print(everyRec)
第10行
第11行 dbRec=dbConn.execute("""select tblBook.ISBN,bookName,tblStudent.studID,Name,borrowDate,returnDate
第12行 from tblBook inner join tblBorrow on tblBook.ISBN=tblBorrow.ISBN
第13行 join tblStudent on tblStudent.studID=tblBorrow.studID
第14行 where returnDate is not null""")
第15行
第16行 print("============")
第17行 for everyRec in dbRec:
第18行     print(everyRec)
第19行
第20行 dbRec.close() #关闭记录集
第21行 dbConn.close() #关闭数据库连接，也就关闭了数据库
```

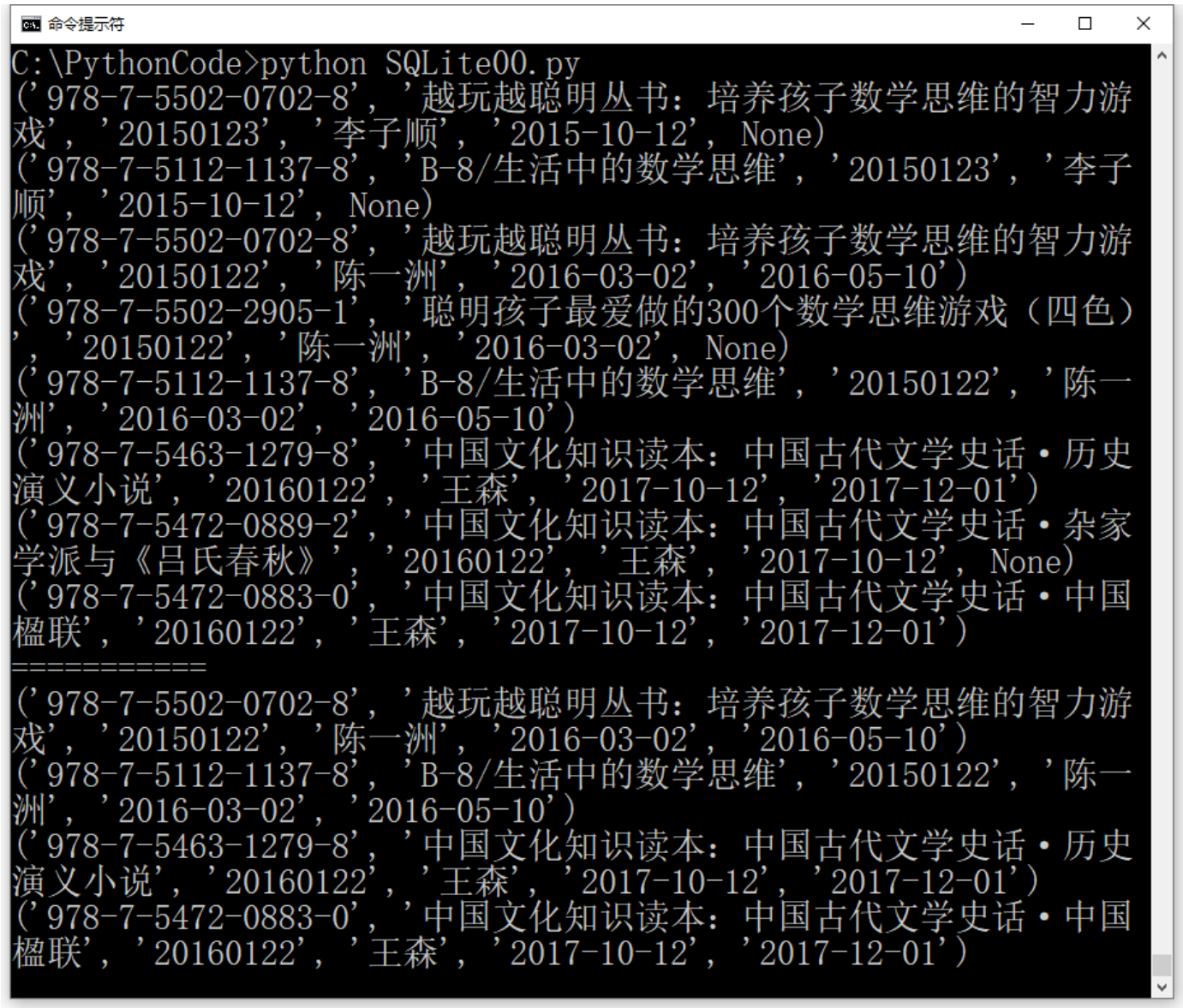

图5-3　例程5-5执行结果

例程第5行的功能是连接到数据库。在SQLite中，如果链接的数据库不存在，则将创建数据库文件，打开文件夹能看到该文件。如果该数据库文件已经存在，则打开文件，等待操作。例程第8-15行创建表，一个数据库可以多个表。dbConn.execute()的参数称之为SQL语句，其中create table的功能是创建表。在例程中，studForm是表的名称。创建表时，要指定表的列名称以及数据类型，如name varchar(10)表示列的名称是name，varchar是数据类型，用于存储字符，10在此处是指定字符串最大长度，即最多装10个字符。在例程create table命令中，多处使用not null，其含义是“不能为空”。当增加数据时，如果没有该列数据，将不允许增加新数据。第9行primary key的含义是“主键”，即当一条一条数据存储到表时，将自动按此排序。如果按主键查找数据，速度很快。

例程5-6

```
第1行  import sqlite3 #使用SQLite的前提
第2行
第3行  #使用数据库第一步是连接到数据库，此处连接SQLite
第4行  #如果文件不存在，会自动在当前目录创建:
第5行  dbConn = sqlite3.connect('student.db') #数据库文件是student.db
第6行
第7行  #执行一条SQL语句，创建studForm表
第8行  dbConn.execute('''create table studForm
第9行          (studID varchar(10) primary key not null,
第10行         name varchar(10) not null,
第11行         id varchar(18) not null,
```

```
第12行             department varchar(20) not null,
第13行             sex boolean not null,
第14行             native varchar(50) not null,
第15行             connection varchar(60) not null)''')
第16行
第17行
第18行    # 继续执行一条SQL语句，插入一条记录:
第19行    dbConn.execute('''insert into studForm
第20行        (studID,id,name,department,sex,native,connection)
第21行        values('1500012123','110108199012281221','吴慈仁','历史系',1,
第22行        '北京宣武','123XXX99912')
第23行        ''')
第24行
第25行    dbConn.execute('''insert into studForm
第26行        (studID,id,name,department,sex,native,connection)
第27行        values('1500012124','440221199512281221','尤义','计算机系',0,
第28行        '广东韶关','127XXX99912')
第29行        ''')
第30行
第31行    dbConn.execute('''insert into studForm
第32行        (studID,id,name,department,sex,native,connection)
第33行        values('1600012124','513030199610282221','黄迪','商学院',1,
第34行        '四川达州','129XXX99912')
第35行        ''')
第36行
第37行    dbConn.commit() #提交事务
第38行
第39行    dbConn.execute(""" update studForm set name='吴迪' where studID='1600012124' """)
第40行    dbConn.commit() #提交事务
第41行
第42行    #删除数据，注意where子句
第43行    dbConn.execute(""" delete from studForm where ID='110108199012281221' """)
第44行    dbConn.commit() #提交事务
第45行
第46行    myCursor = dbConn.execute("select studID,name,department from studForm")
第47行    for row in myCursor:
第48行        print("学号："+row[0]+" 姓名："+row[1]+" 院系："+row[2])
第49行
第50行    myCursor.close() #关闭Cursor
第51行
第52行    dbConn.close() # 关闭Connection
第53行
第54行    #eof
```

```
学号：1500012124    姓名：尤义    院系：计算机系
学号：1600012124    姓名：吴迪    院系：商学院
```

图5-4　例程5-6执行效果

第18-35行用于增加三条数据，其SQL命令是insert into，其后跟数据插入的表格名称，此处为studForm。括号中指明插入数据的列名称，values括号中指定对应的数据。commit()用于确保多个数据连接一致性，当前的数据修改等反应到其他数据库连接。

第39行update命令用于修改数据，where子句用于限定条件。如果省略where子句，则将修改所有指定列的数据。此处限定修改studID等于1600012124的记录。

第43行delete命名用于删除记录，同样，where子句用于限定条件。当省略where子句时，将删除指定表的全部数据。

第46行select命令用于选择记录，可以跟随where子句限定条件选择某些记录。select可以限定列，select * from studForm则表示选中所有列所有记录。第47行提供了逐行访问记录的方式，显示效果如图5-4所示。

数据库一般都支持多表操作，SQLite也不例外。例程5-7在例程5-6的基础上增建studScore，用于保存学生成绩，字段名称分别是：studID(学号)、Math(数学)、Culture(语文)、Foreign(外语)。

例程5-7

```
第1行  import sqlite3 #使用SQLite的前提
第2行
第3行  #student.db已经存在studForm表，在此基础上增加studScore表
第4行  dbConn = sqlite3.connect('student.db') #数据库文件是student.db
第5行
第6行  #执行一条SQL语句，创建studScore表
第7行  dbConn.execute('''create table studScore
第8行          (studID varchar(10) primary key not null,
第9行          Math real not null,Culture real not null,Language int not null)''')
第10行
第11行
第12行 # 继续执行一条SQL语句，插入一条记录:
第13行 dbConn.execute('''insert into studScore
第14行    (studID,Math,Culture,Language)
第15行    values('1500012123',98,87,90)''')
第16行
第17行 dbConn.execute('''insert into studScore
第18行    (studID,Math,Culture,Language)
第19行    values('1500012124',65,90,80)    ''')
第20行
第21行 dbConn.execute('''insert into studScore
第22行    (studID,Math,Culture,Language)
第23行    values('1600012124',85,85,90)    ''')
第24行
第25行 dbConn.commit() #提交事务
第26行
第27行 myCursor = dbConn.execute("""select studForm.studID,name,department,Math,Culture,Language from stud
       Form,studScore
第28行 where studForm.studID=studScore.studID""")
第29行
第30行 for row in myCursor:
第31行    print("学号："+row[0]+" 姓名："+row[1]+" 院系："+\
第32行    row[2]+" 数学："+str(row[3])+" 语文："+str(row[4])+" 外语："+str(row[5]))
第33行
第34行 myCursor = dbConn.execute("""select studForm.studID,name,Math,Culture,Language,
```

```
第35行    Math+Culture+Language as totalScore,(Math+Culture+Language)/3 from studForm,studScore
第36行    where studForm.studID=studScore.studID""")
第37行
第38行    for row in myCursor:
第39行        print("学号："+row[0]+" 姓名："+row[1]+" 数学："+\
第40行        str(row[2])+" 语文："+str(row[3])+" 外语："+str(row[4])+\
第41行        " 总成绩："+str(row[5])+" 平均成绩："+'{:02.2f}'.format(row[6]))
第42行
第43行    myCursor.close() #关闭Cursor
第44行
第45行    dbConn.close() # 关闭Connection
第46行
第47行    #eof
```

```
学号：1500012124   姓名：尤义   院系：计算机系   数学：65.0   语文：90.0   外语：80
学号：1600012124   姓名：吴迪   院系：商学院   数学：85.0   语文：85.0   外语：90
学号：1500012124   姓名：尤义   数学：65.0   语文：90.0   外语：80   总成绩：235.0   平均成绩：78.33
学号：1600012124   姓名：吴迪   数学：85.0   语文：85.0   外语：90   总成绩：260.0   平均成绩：86.67
```

图5-5　例程5-7执行效果

从表的建立和数据插入可以看出，studScore中只有学号和成绩没有学生姓名等的信息，如果查询学生成绩，很明显不能显示姓名等信息将很不友好。studID在两个表中都存在，只要建立起两者之间的关系，就可以获得相关信息，如例程第27-28行所示，其中from子句有两个表的名称，where子句建立起两个连接的条件，即两个表的studID相等。

第27-28行的studID前面有studForm或studScore，其功能是指明studID来源。当字段名称可能来自多个表时，必须指明其所属关系。

观察第34-36行代码可以发现，字段可以做运算如Math+Culture+Language as totalScore是将Math、Culture和Language相加作为一个新字段并将其命名为totalScore。新字段可以命名，也可以不命名如其后的(Math+Culture+Language)/3就没有命名。在设计数据库表结构时，一般说来不能存在类似总成绩、平均成绩这样的字段，这些字段的值可以从基本字段经过运算得到。如果有这样的字段存在，修改数据时将多处修改，难以保持一致性。在数据库设计中，这条原则称之为数据原子性，即数据不可再分。对于总成绩，可以分解为Math、Culture和Language所以不能存在。

对于常见的查询，可以在数据库中命名保存，称之为试图(view)，以备各种场景的应用，如例程5-8所示。视图与表的使用几乎没有区别。

例程5-8

```
第1行    import sqlite3 #使用SQLite的前提
第2行
第3行    dbConn = sqlite3.connect('student.db') #数据库文件是student.db
第4行
第5行    dbConn.execute("""create view scoreList as
第6行    select studForm.studID,name,department,Math,Culture,Language,
第7行    (Math+Culture+Language) as totalScore,(Math+Culture+Language)/3 as avgScore
第8行    from studForm,studScore where studForm.studID=studScore.studID""")
第9行
第10行   dbConn.commit()
第11行
第12行   myCursor=dbConn.execute("select studID,name,totalScore,avgScore from scoreList")
第13行   for row in myCursor:
第14行       print("学号："+row[0]+" 姓名："+row[1]+" 总成绩："+str(row[2])+" 平均成绩："+"{:2.2f}".format(row[3]))
第15行
第16行
```

```
myCursor.close() #关闭Cursor
第17行
第18行 dbConn.close() # 关闭Connection
第19行
第20行 #eof
```

例程5-9第5行从视图scoreList中选出name、Math、Culture、Language共4列，其执行效果如图5-6所示。从中可以看出，记录总体表现为list，每个记录为tuple形式。这是第6行代码执行结果，当在cursor中执行fetchall()时，每个记录为tuple，总体总计为list形式，这为数据操作带来更大灵活性。

例程5-9

```
第1行 import sqlite3 #使用SQLite的前提
第2行
第3行 dbConn = sqlite3.connect('student.db') #数据库文件是student.db
第4行
第5行 myCursor=dbConn.execute("select name,Math,Culture,Language from scoreList")#从视图中选择
第6行 print(myCursor.fetchall())
第7行
第8行 myCursor.close() #关闭Cursor
第9行
第10行 dbConn.close() # 关闭Connection
第11行
第12行 #eof
```

```
D:\myBook\Python\Python_Prog>python db02.py
[('尤义', 65.0, 90.0, 80), ('吴迪', 85.0, 85.0, 90)]
```

图5-6　例程5-9执行效果

第二节　SQL进阶

SQL是专门语言，用于在数据库中存取数据以及查询、更新和管理等等。不同底层结构不同运行机制的不同数据库系统，几乎可以使用相同的SQL语句。1986年10月，ANSI对SQL进行规范，以作为关系式数据库管理系统(Relational DataBase Mangement System，简称RDBMS)的标准语言，1987年成为国际标准。虽然各种通行数据库在其实践过程中，对标准SQL都或多或少进行扩充，但基本相差不大，熟悉某种数据库的SQL可以较为轻松地适应其他数据库。

根据SQL语句的功能，可将其分为六种。

- **数据查询语句(Data Query Language，简称DQL)：**可以说就是select语句，也称为“数据检索语句”，用以从单表或多表中检索数据，常与where条件子句，order by排序子句，group by分组子句以及having分组过滤子句。
- **数据定义语句(Data Definition Language，简称DDL)：**包括：create table(创建表)、create view(创建视图)和create index(创建索引)以及drop table(删除表)、drop index(删除索引)和drop view(删除视图)等。另外，还有alter table用于修改已经创建的表。创建表或视图时，该表或视图不能已经存在。删除表和视图时，务必谨慎，一旦删除数据很难甚至不能恢复。
- **数据操作语句(Data Manipulation Language，简称DML)：**包括：insert、update以及delete，分别用于添加、修改和删除表中的行。这三种语句，将导致表中数据发生改变。对于update和delete，一般都有where子句限定更新或删除条件，否则将导致全表数据指定列被更新或全表被删除。
- **游标控制语句(Cursor Control Language，简称CCL)：**如本章前述例程所用到的cursor即是游标控制语句，用于对选择语句执行结果的行操作。cursor如同指针，可以依次或跳跃指向相关行。
- **事务处理语句(Transaction Procession Language，简称TPL)：**DPL用于确保被DML语句影响的表的所有行及时得以更新。TPL语句包括begin transaction(事务启动)，commit(提交)和rollback(回滚)等。
- **数据控制语句(Data Control Language，简称DCL)：**对于多用户数据库，可通过DCL语句对用户授权访问数据库或数据库的表或视图等，包括grant以及revoke语句。某些RDBMS甚至可以用grant或revoke语句对列访问授权，使得控制粒度更小。SQLite是较为简单的数据库，没有DCL语句。

1、查询语句

在SQL中，查询语句就是select语句。其基本语法格式如下，其方括号内为可选项，可单独使用也可联合使用。如果字段名称位置为*，则表示将表名称所代表的表中所有字段均列出。为了更好地练习SQL语句，可以在前述例程所涉及表中增加更多数据以便于观察效果，例程5-10是select语句的一些应用示例。

select 单或多字段名称 from 表名称 [where|group by [having]|order by]

例程5-10第6行代码select * from studForm表示选出studForm所有记录的所有列；第9-10行则表示选出studForm和studScore中的所有列，studForm.*表示studForm中的所有列，studScore.*与之相似。由于第-10行涉及到两个表，因此要定义这两个表连接的方式，其where子句表明两表需studID相同。

例程5-10

```
第1行  import sqlite3 #使用SQLite的前提
第2行
第3行  #student.db已经存在studForm表，在此基础上增加studScore表
第4行  dbConn = sqlite3.connect('student.db') #数据库文件是student.db
第5行
第6行  myCursor=dbConn.execute("select * from studForm")#选出studForm中所有列
第7行  print(myCursor.fetchall())
第8行
第9行  myCursor=dbConn.execute("""select studForm.*,studScore.* from studForm,studScore
第10行 where studForm.studID=studScore.studID""") #选出studForm和studScore中所有列
第11行 print(myCursor.fetchall())
第12行
第13行 myCursor=dbConn.execute("""select studForm.studID as 学号,name as 姓名,Math,Culture,Language
第14行 from studForm join studScore on studForm.studID=studScore.studID
第15行 where Math>80 """) #注意inner join的使用
第16行 print(myCursor.fetchall())
第17行
第18行 myCursor=dbConn.execute("""select studForm.studID,name,Math,Culture,Language
第19行 from studForm join studScore using(studID) where Math>60 """) #注意using的使用
第20行 print(myCursor.fetchall())
第21行
第22行 myCursor=dbConn.execute("""select studForm.studID,name,Math,Culture,Language
第23行 from studForm cross join studScore """) #cross join
第24行 print(myCursor.fetchall())
第25行
第26行 myCursor=dbConn.execute("""select studForm.studID,name,Math,Culture,Language
第27行 from studScore left join studForm using(studID) """) #left join
第28行 print(myCursor.fetchall())
第29行
第30行 myCursor.close() #关闭Cursor
第31行 dbConn.close() # 关闭Connection
第32行
第33行 #eof
```

第13-15行的SQL语句中出现了as子句，用于设定别名，如“studForm.studID as 学号”将studForm中的studID命名为“学号”。from子句中出现了join，其功能是定义两个或多个表的连接方式。where子句用于设置记录过滤条件。表的连接方式与记录

过滤条件可以放在一起，但分开更加自然友好易于理解。如果连接的两个表有相同名称的字段，可以用using子句，代码更精简易于理解，如例程第18-19行所示。

表与表之间的连接可以分为交叉连接(cross join)，内连接(inner join)和外连接(outer join)，默认为内连接，此时可以省略inner。交叉连接将形成两个或多个表的笛卡尔集，如A表有5条记录，B表有10条记录，将形成5×10共50条记录，或者说相当于A表的第一条记录与B表的每一条记录无条件组合，然后第二条记录与B表的每一条记录无条件组合，以此类推。cross join是两个或多个表能形成的最大记录数，其他连接方式所产生的记录数都是其子集。例程第22-23行是cross join。cross join同样可以有where子句等。

SQL标准外链接还可以分为left outer join、right outer join以及full outer join，但SQLite仅支持left outer join。此处仅讲解left outer join。当使用外链接时，可以省略outer为left join、right join和full join，例程26-27行为left join示例。left join的含义是以左连接的表为准，如果右边表与左边表记录匹配，显示两个表的信息，否则，显示为None。在例程第26-27行中，如果左边表studForm有3条记录，其中有两条记录的studID与右边表studScore相同，则两条相同记录与inner join相同，而不同的记录则右边表的信息部分显示为None，在有些数据库中显示null。

```
[('1500012124', '尤义', 98.0, 87.0, 90), ('1500012124', '尤义', 65.0, 90.0, 80), ('1500012124'
, '尤义', 85.0, 85.0, 90), ('1600012124', '吴迪', 98.0, 87.0, 90), ('1600012124', '吴迪', 65.0
, 90.0, 80), ('1600012124', '吴迪', 85.0, 85.0, 90)]
[(None, None, 98.0, 87.0, 90), ('1500012124', '尤义', 65.0, 90.0, 80), ('1600012124', '吴迪',
85.0, 85.0, 90)]
```

图5-7　例程5-10中cross join和left join执行效果

order by子句用于设定升序或降序，可以设置单列或多列。假定表5-1是某公司薪酬表，数据存储company.db的salary表中，例程5-11第6行order by的使用，此处“工资”按升序，“年龄”按降序。当按照升序排序时，可以省略asc，此为默认值

表5-1：某公司薪酬表

序号	姓名	性别	年龄	籍贯	工资
1001	张珊	女	32	四川	8000.00
1002	李思	女	37	广东	6000.00
1003	王武	男	28	广东	7500.00
1004	刘浏	男	40	湖南	8000.00
1005	齐琪	女	24	湖南	6000.00
1006	巴山	男	45	四川	10000.00

例程5-11

```
第1行  import sqlite3 #使用SQLite的前提
第2行
第3行  #student.db已经存在studForm表，在此基础上增加studScore表
第4行  dbConn = sqlite3.connect('company.db') #数据库文件是student.db
第5行
第6行  myCursor=dbConn.execute("select * from salary order by 工资 asc,年龄 desc ")
第7行  print(myCursor.fetchall())
第8行
第9行  myCursor=dbConn.execute("select 性别,sum(工资) as 总额 from salary group by 性别 order by 总额 desc")
第10行 print(myCursor.fetchall())
第11行
第12行 myCursor=dbConn.execute(""" select 籍贯,count(*) as 人数,sum(工资) as 总额,min(工资) as 最低工资,
第13行     max(工资) as 最高工资,avg(工资) as 平均工资 from salary group by 籍贯 order by 总额 desc""")
第14行 print(myCursor.fetchall())#group by必须放在where之后，order by之前
第15行
第16行 myCursor=dbConn.execute(""" select 年龄-年龄%10 as 年纪,sum(工资) as 总额 from salary
第17行 group by 年纪 having 总额>13800 """)
```

```
print(myCursor.fetchall())

myCursor.close() #关闭Cursor
dbConn.close() # 关闭Connection

#eof
```

```
[('1002', '李思', '女', 37, '广东', 6000.0), ('1005', '齐琪', '女', 24, '湖南', 6000.0),
('1003', '王武', '男', 28, '广东', 7500.0), ('1004', '刘浏', '男', 40, '湖南', 8000.0), (
'1001', '张珊', '女', 32, '四川', 8000.0), ('1006', '巴山', '男', 45, '四川', 10000.0)]
[('男', 25500.0), ('女', 20000.0)]
[('四川', 2, 18000.0, 8000.0, 10000.0, 9000.0), ('湖南', 2, 14000.0, 6000.0, 8000.0, 7000
.0), ('广东', 2, 13500.0, 6000.0, 7500.0, 6750.0)]
[(30, 14000.0), (40, 18000.0)]
```

图5-8　例程5-11中执行效果

group by用于分组，如例程第9行按性别分组，分别统计男性和女性的工资总额，对统计结果按总额高低降序排列。sum()称之为汇总函数，除了sum()外，还有其他汇总函数，如例程第12-13行所示。sum()用于汇总、min()求最小值、max()求最大值、avg()求平均值、count()统计个数。

group by还可以有having子句，用于设定汇总后显示条件。在SQL语句中，group by 必须在where之后，order by之前。另外，where子句用于汇总前的记录筛选过滤，而having则是对汇总后的数据进行过滤筛选。这是where和having的重要区别。

where子句变化较多，不但可以应用于select语句，也可以用在update、delete语句。例程5-12是where子句应用示例。在第6行代码where 性别='男' and 工资>6000中，有关系运算符等号(=)和逻辑运算符大于(>)以及逻辑运算符and。表5-2列出了SQLite的各种运算符。SQL运算较多，但都还比较容易理解。

例程5-12

```
import sqlite3 #使用SQLite的前提

#student.db已经存在studForm表，在此基础上增加studScore表
dbConn = sqlite3.connect('company.db') #数据库文件是student.db

myCursor=dbConn.execute("select * from salary where 性别='男' and 工资>6000 ")
print(myCursor.fetchall())

myCursor=dbConn.execute("select * from salary where 序号 in('1001','1005') ")
print(myCursor.fetchall())

myCursor=dbConn.execute("select * from salary where 序号 not in('1001','1005') ")
print(myCursor.fetchall())

myCursor=dbConn.execute("select * from salary where 工资 between 6000 and 10000 ")
print(myCursor.fetchall())

print("here!")
myCursor=dbConn.execute("select * from salary where 序号 like '100[235]' ")
print(myCursor.fetchall())#全部全部记录
print("here!")

myCursor=dbConn.execute("select * from salary where 序号 in(select 序号 from salary where 工资>6000) ")
print(myCursor.fetchall())
```

```
第25行
第26行  myCursor=dbConn.execute("select * from salary where exists(select 序号 from salary where 工资>8000) ")
第27行  print(myCursor.fetchall())#列出全部记录
第28行
第29行  myCursor=dbConn.execute("select * from salary where exists(select 序号 from salary where 工资>10000) ")
第30行  print(myCursor.fetchall())#无记录被列出
第31行
第32行  myCursor=dbConn.execute("select distinct 工资 from salary")
第33行  print(myCursor.fetchall())#无记录被列出
第34行
第35行  myCursor.close() #关闭Cursor
第36行  dbConn.close() # 关闭Connection
第37行
第38行  #eof
```

where子句中，in和not in很常用，如例程第9行、12行以及21行。当使用in时，in前的字段名与in括号内值列表进行比较，如果匹配则列出，否则不列出，not in与之相反。值列表可以逐一列出，也可以用子查询列出，与值列表功能相同。exists和not exists与in和not in用法相似，但有本质区别。如第24行、27行，如果子查询有超过一条记录列出，则主查询的记录全部列出，否则一条记录也不列出。

在like和not like中可以使用通配符%和_，其中%代表1个或多个字符，_仅代表一个字符，如例程第18行所示。GLOB与like用法相似，不过like不区分大小写，而GLOB则区分大小写。

select语句还可以与关键字distinct一起使用，相同记录仅保留一条，如例程第32行所示。

表5-2：SQLite运算符

运算符	描述
	算术运算符
+	加法 - 把运算符两边的值相加
-	减法 - 左操作数减去右操作数
*	乘法 - 把运算符两边的值相乘
/	除法 - 左操作数除以右操作数
%	取模 - 左操作数除以右操作数后得到的余数
	关系运算符
==	检查两个操作数的值是否相等，如果相等则条件为真。
=	检查两个操作数的值是否相等，如果相等则条件为真，与==功能相同。
!=	检查两个操作数的值是否相等，如果不相等则条件为真。
<>	检查两个操作数的值是否相等，如果不相等则条件为真，与!=功能相同。
>	检查左操作数的值是否大于右操作数的值，如果是则条件为真。
<	检查左操作数的值是否小于右操作数的值，如果是则条件为真。
>=	检查左操作数的值是否大于等于右操作数的值，如果是则条件为真。
<=	检查左操作数的值是否小于等于右操作数的值，如果是则条件为真。
!<	检查左操作数的值是否不小于右操作数的值，如果是则条件为真，相当于大于等于。
!>	检查左操作数的值是否不大于右操作数的值，如果是则条件为真，相当于小于等于。
	逻辑运算符
AND	AND 运算符允许在一个 SQL 语句的 WHERE 子句中的多个条件的存在。
NOT	NOT 运算符是所用的逻辑运算符的对立面。比如 NOT EXISTS、NOT BETWEEN、NOT IN，等

	等。**它是否定运算符。**
OR	OR 运算符用于结合一个 SQL 语句的 WHERE 子句中的多个条件。
BETWEEN	BETWEEN 运算符用于在给定最小值和最大值范围内的一系列值中搜索值。
EXISTS	EXISTS 运算符用于在满足一定条件的指定表中搜索行的存在。
IN	IN 运算符用于把某个值与一系列指定列表的值进行比较。
NOT IN	IN 运算符的对立面，用于把某个值与不在一系列指定列表的值进行比较。
LIKE	LIKE 运算符用于把某个值与使用通配符运算符的相似值进行比较。
GLOB	GLOB 运算符用于把某个值与使用通配符运算符的相似值进行比较。GLOB 与 LIKE 不同之处在于，它是大小写敏感的。
IS NULL	NULL 运算符用于把某个值与 NULL 值进行比较。
IS	IS 运算符与 = 相似。
IS NOT	IS NOT 运算符与 != 相似。
||	连接两个不同的字符串，得到一个新的字符串。
UNIQUE	UNIQUE 运算符搜索指定表中的每一行，确保唯一性（无重复）。

2、定义语句

定义语句包括：create table(创建表)、create view(创建视图)和create index(创建索引)以及drop table(删除表)、drop view(删除视图)以及drop index(删除索引)。

(1) create table

create table用于创建表，在关系数据库中，主要有两种创建方式，可以称之为CREATE TABLE方式，也是主要的方式。例程5-6是一种方式，例程5-13第7行是另外一种方式，可以称之为CREATE TABLE AS方式，即根据as后select语句执行结果创建新的表结构，同时包括其数据。

例程5-13

```
第1行  import sqlite3 #使用SQLite的前提
第2行
第3行  #student.db已经存在studForm表，在此基础上增加studScore表
第4行  dbConn = sqlite3.connect('company.db') #数据库文件是student.db
第5行
第6行  #create table as示例
第7行  dbConn.execute("create table tblTmp as select 序号,姓名,工资 from salary;")
第8行  dbConn.commit()
第9行
第10行 myCursor=dbConn.execute("select * from tblTmp;")
第11行 print(myCursor.fetchall())
第12行
第13行 myCursor.close() #关闭Cursor
第14行 dbConn.close() # 关闭Connection
第15行
第16行 #eof
```

在数据库中创建表，必须指定表名称、字段名称、字段数据类型，除此以外，还可以设定字段的其他属性，如：主键(primary key)、默认值(default value)、排序规则(collation)、非空(not null)、唯一性(unique)、条件约束(check)、自增(auto increment)以及外键(foreign key)。

在例程5-6中，studID为主键(primary key)且其值不能为空(not null)。主键是行或者记录的唯一标识，且一个表有且仅能有为一个主键。主键可以由一个或多个字段组成。主键不可重复，也不可以为空。

(2) alert table

(3) drop table

已创建数据表可以用drop table命令删除，其语法如下，其中tableName是数据表名称。

```
drop table tableName
```

(4) create view

视图可以称之为虚拟表或逻辑表，也可以说是对查询语句命名并存储，其操作类似真实表(也称物理表)。同真实表一样，视图由行列组成。但视图数据并不真实存储在视图之中，而是在引用视图时动态生成，这是和真实表最大的不同。create view的语法如下，其中viewName是视图的名称，select-statement是select语句。当视图建立，可以如table一样使用，如例程5-14第4、7行所示。

```
create view viewName as select-statement
```

例程5-14

```
第1行 import sqlite3 #使用SQLite的前提
第2行
第3行 dbConn = sqlite3.connect('company.db')
第4行 dbConn.execute("create view briefData as select 序号,姓名,性别,工资 from salary")
第5行 dbConn.commit()
第6行
第7行 myCursor=dbConn.execute("select * from briefData where 工资>=7000")
第8行 print(myCursor.fetchall())
第9行 dbConn.close() # 关闭Connection
第10行
第11行 #eof
```

建立视图的select语句在视图创建过程中一般会进行预处理(类似高级语言之编译)，因此，其执行效率通常比直接使用select语句高，尤其是数据量较大时更加明显。

(5) drop view

已创建视图可以用drop view命令删除，其语法如下，其中viewName是视图名称。

```
drop view viewName
```

(6) create index

索引是对数据库中表的一个或多个列的值进行预先排序(在数据更新时同时进行排序)。如查询：select * from salary where 序号="1005"。如果没有索引，必须遍历整个表，直到"序号"等于"1005"为止；如果有索引之后(必须是在"序号"这一列建索引)，即可先在索引中查找并找到对应表对应行。由于索引及其查找算法都经过大量优化，因此查找次数大大减少，大大提升效率。建立良好的索引，可大大优化数据查询效率。创建索引的命令如下，其中indexName是新创建索引的名称，tableName是索引所在的数据表。索引依赖数据表而存在。columnNameList是列名称列表，可以是一个字段或多个字段。unique是可选项，当选择unique参数时，则索引值不能重复，如在salary表中，"序号"不能重复，如果建立基于"序号"的索引，则可以加上unique选项，如果建立"工资"的索引，则不能有unique选项。相对来讲，unique选项效率较高。如果某列值肯定不会重复，很有必要使用unique选项。另外，创建数据表时可以同时建立其相关索引。

```
create [unique] index indexName on tableName (columNameList)
```

(7) drop index

删除索引的语法如下，其中indexName是索引名称。

```
drop index indexName
```

3、操作语句

数据操作语句包括：insert、update以及delete，分别用于添加、修改和删除表中的行。当使用update和delete语句时，可以默认为必须有where子句，否则将导致所有行的数据被更新或者删除。

(1) insert

insert语句用于向数据库的表存入数据，其格式如下，其中tableName是表的名称，colNameList是一个或多个列名称，valueList是值的序列，其个数与colNameList匹配。当使用“格式2”时，可以把已在其他表中(一般不是同表)的数据插入到表中。例程5-15有insert语句示例。

```
格式1：insert into tableName (colNameList) values (valueList)
格式2：insert into tableName (select colNameList from tableName)
```

(2) update

例程5-15有insert语句示例。

```
update tableName set colName1=value1,colName2=value2,..... [where 子句]
```

(3) delete

例程5-15有insert语句示例。

```
delete from tableName [where 子句]
```

例程5-15

```
第1行 import sqlite3 #使用SQLite的前提
第2行
第3行 dbConn = sqlite3.connect('company.db')
第4行 #dbConn.execute("insert into salary(序号,姓名,性别,年龄,籍贯,工资) values('{}','{}','{}',{},'{}',{})".format("1009","单杉杉","女",42,"山西",8000))
第5行 dbConn.commit()
第6行 print(dbConn.execute("select * from salary").fetchall())
第7行
第8行 dbConn.execute("update salary set 工资=工资*1.2") #所有人工资上调20%
第9行 dbConn.commit()
第10行 print(dbConn.execute("select * from salary").fetchall())
第11行
第12行 #带有where子句
第13行 dbConn.execute("update salary set 工资=工资*1.2 where 姓名='巴山'") #仅'巴山'工资上调20%
第14行 dbConn.commit()
第15行 print(dbConn.execute("select * from salary").fetchall())
第16行
第17行 #如果删除where子句，则将删除salary中的所有数据
第18行 dbConn.execute("delete from salary where 姓名='巴山'") #删除'巴山'的所在行
第19行 dbConn.commit()
第20行 print(dbConn.execute("select * from salary").fetchall())
第21行
```

```
第22行
第23行  dbConn.close() # 关闭Connection
第24行
第25行  #eof
```

4、游标语句

5、事务语句

某些工作常常需要执行一系列操作，这些操作要么完全执行要么完全不执行。如“A账户向B账户汇款500元”，需要执行的操作大致分为“①读出A账户余额(假如1200元)→②A账户余额减去500元(1200-500=700元)为700元→③读出B账户余额(假如400元)→④B账户余额加上500元变更为900元”。上述操作可能在执行过程中发生意外，比如B账户突然被注销、系统故障等等。数据库设计必须保证即便发生类似情况也能保证操作要么全部被执行要么全部不执行，这在数据库中被称为事务(Transaction)，即单个逻辑工作单元执行的一系列操作，要么完全执行，要么完全不执行。数据库事务处理可以确保除非事务单元内的所有操作都成功完成，否则不会永久更新面向数据的资源。一个逻辑工作单元要成为事务，必须满足所谓ACID（原子性、一致性、隔离性和持久性）原则，更多了解可阅读数据库相关书籍。

在SQLite3中，事务语句包括：BEGIN/BEGIN TRANSACTION(启动事务)、ROLLBACK(回滚)、COMMIT/END TRANSACTION(提交或结束事务)。例程5-16是有关事务的示例。

例程5-16

```
第1行   import sqlite3 #使用SQLite的前提
第2行
第3行   #判断N是否为质数---开始
第4行   def isPrime(N):
第5行       if N<=1:return "否"
第6行       if N==2:return "是"
第7行       for i in range(2,N):
第8行           if N%i==0:return "否"
第9行       return "是"
第10行  #判断N是否为质数---结束
第11行
第12行  dbConn=sqlite3.connect(":memory:") #创建内存数据库
第13行  dbConn.execute("create table tabPrime (intNum,isPrime)") #用来保存一个数是否质数
第14行
第15行  print(dbConn.execute("select * from tabPrime").fetchone())#输出：None
第16行
第17行  dbConn.execute("BEGIN") #亦可将BEGIN改为BEGIN TRANSACTION
第18行  for i in range(1,10000):
第19行      dbConn.execute("insert into tabPrime(intNum,isPrime) values(?,?)",(i,isPrime(i)))
第20行  print(dbConn.execute("select * from tabPrime").fetchone()) #输出：(1, '否')
第21行
第22行  dbConn.rollback() #回滚
第23行  print(dbConn.execute("select * from tabPrime").fetchone()) #输出：None
第24行
第25行  dbConn.close()
第26行
第27行  #eof
```

6、触发器

当执行insert/update/delete时，可以自动执行预先设定的相关SQL语句，此为触发器(Trigger)，例程5-17是触发器的简单示例。

例程5-17

```
第1行  import sqlite3 #使用SQLite的前提
第2行
第3行  dbConn = sqlite3.connect('bank.db') #数据库文件是bank.db
第4行  dbConn.execute("""create table tabMain (
第5行      客户编号 char(8) primary key not null,
第6行      客户姓名 varchar(20),
第7行      账户余额 real,
第8行      操作日期 integer
第9行      )""")
第10行
第11行 dbConn.execute("""create table tabDetail (
第12行     操作序号 integer primary key autoincrement,
第13行     客户编号 char(8) not null,
第14行     本次金额 real,
第15行     本次日期 integer
第16行     )""")
第17行
第18行 dbConn.execute("""CREATE TRIGGER newUser AFTER INSERT ON tabMain
第19行 BEGIN
第20行    INSERT INTO tabDetail(客户编号,本次金额,本次日期) VALUES (new.客户编号,new.账户余额,datetime('now'));
第21行 END;""") #当在tabMain执行insert后，将在tabDetail执行的操作
第22行
第23行 dbConn.execute("""CREATE TRIGGER changeDetail AFTER INSERT ON tabDetail
第24行 BEGIN
第25行    update tabMain set 账户余额=账户余额+new.本次金额,本次日期=new.本次日期
第26行    where 客户编号=new.客户编号;
第27行 END;""") #当在tabDetail执行insert后，将在tabMain执行的操作
第28行
第29行 dbConn.close() # 关闭Connection
第30行
第31行 #eof
```

第三节　connection对象

第四节　cursor对象

第五节　Web实例

例程5-18

```
第1行  #程序文件名：borrow.py
第2行  from flask import Flask,request
第3行  import sqlite3
第4行
第5行  dbConn=sqlite3.connect("lib.db",check_same_thread=False)
```

```
app=Flask("__name__")

@app.route("/")
def Index():
    return app.send_static_file("Index.html")

@app.route("/studID",methods=("post",))
def searchStudentID():
    studID=request.form["studID"]
    myCur=dbConn.execute(f"select studID,Name,Gender from tblStudent where studID='{studID}'")
    allRec=myCur.fetchall()
    myCur.close() #关闭记录集

    if len(allRec)!=1:
        return f'{{"state":0}}'
    else:
        print(allRec)
        return f'{{"state":1,"studID":"{allRec[0][0]}","Name":"{allRec[0][1]}","Gender":"{allRec[0][2]}"}}'

@app.route("/bookName",methods=("post",))
def searchBookName():
    bookName=request.form["bookName"]
    myCur=dbConn.execute(f"select ISBN,bookName,publisher from tblBook where bookName like '%{bookName}%'")
    allRec=myCur.fetchall()
    myCur.close() #关闭记录集

    if len(allRec)==0:
        return f'{{"state":0}}'
    else:
        print(allRec)
        content=""
        for everyRec in allRec:
            content+=f'{{"ISBN":"{everyRec[0]}","bookName":"{everyRec[1]}","publisher":"{everyRec[2]}"}},'
        else:
            content=content[:-1]
        content=f'{{"state":1,"bookList":[{content}]}}'
        print(content)
        return content

@app.route("/Borrow",methods=("post",))
def borrowBook():
    studID=request.form["studID"]
    isbnList=request.form["isbnList"]
    isbnList=isbnList.split(",")
```

```
    borrowDate=request.form["borrowDate"]
    for everyISBN in isbnList:
        strSQL=f"insert into tblBorrow (studID,isbn,borrowDate) values('{studID}','{everyISBN}','{borrowDate}')"
        dbConn.execute(strSQL)
    dbConn.commit()
    print(studID,isbnList,borrowDate)

    return "ABC"

@app.route("/studID_Borrow",methods=("post",))
def searchStudentID_Borrow():
    studID=request.form["studID"]
    myCur=dbConn.execute(f"select studID,Name,Gender from tblStudent where studID='{studID}'")
    allRec=myCur.fetchall()
    myCur.close() #关闭记录集

    if len(allRec)!=1:
        studData=f'{{"state":0}}'
        return studData
    else:
        print(allRec)
        studData=f'"state":1,"studID":"{allRec[0][0]}","Name":"{allRec[0][1]}","Gender":"{allRec[0][2]}"'

        #取得尚未还书的信息
        strSQL=f"select tblBorrow.borrowID,tblBorrow.ISBN,tblBook.bookName,tblBook.publisher,tblBorrow.borrowDate from tblBorrow inner join tblBook on tblBorrow.ISBN=tblBook.ISBN where tblBorrow.studID='{studID}' and tblBorrow.returnDate is null"
        print(strSQL)
        myCur=dbConn.execute(strSQL)
        allRec=myCur.fetchall()
        myCur.close()
        notReturnAmount=len(allRec)

        if notReturnAmount==0:
            studData+=f',"notReturnAmount":0'
        else:
            bookList=""
            for everyRec in allRec:
                bookList+=f'{{"borrowID":"{everyRec[0]}","ISBN":"{everyRec[1]}","bookName":"{everyRec[2]}","publisher":"{everyRec[3]}","borrowDate":"{everyRec[4]}"}},'
            else:
                bookList=bookList[:-1]
            studData+=f',"notReturnAount":{notReturnAmount},"bookList":[{bookList}]'
        print(studData)
    return f'{{{studData}}}'
```

```
第92行
第93行  @app.route("/Return",methods=("post",))
第94行  def returnBook():
第95行      studID=request.form["studID"]
第96行      borrowIDList=request.form["borrowIDList"]
第97行      borrowIDList=borrowIDList.split(",")
第98行      returnDate=request.form["returnDate"]
第99行      for everyRtn in borrowIDList:
第100行         strSQL=f"update tblBorrow set returnDate='{returnDate}' where studID='{studID}' and borrowID='{everyRtn}'"
第101行         dbConn.execute(strSQL)
第102行         print(strSQL)
第103行     dbConn.commit()
第104行
第105行     return "ABC"
第106行
第107行
第108行 if __name__=="__main__":
第109行     app.run(port=80,debug=True)
```

例程5-19

```
第1行  <!--static下的Index.html-->
第2行  <!doctype html>
第3行  <html lang="zh-cn">
第4行   <head>
第5行    <meta charset="UTF-8">
第6行    <title>格致中学---图书馆管理系统</title>
第7行     <link rel="stylesheet" type="text/css" href="static/css/style.css"/>
第8行   </head>
第9行   <body style="">
第10行   <div id="main">
第11行       <br><br><div id="gzTitle">格致中学图书馆管理系统</div><br><br>
第12行       <div id="Fx">
第13行           <a href="static/borrow.html">借书</a>
第14行           <a href="static/return.html">还书</a><br><br>
第15行       </div>
第16行    </div>
第17行   </body>
第18行  </html>
```

例程5-20

```
第1行  <!--保存在static下的borrow.html-->
第2行  <!doctype html>
第3行  <html lang="zh-cn">
第4行   <head>
第5行    <meta charset="UTF-8">
```

```
第6行    <title>格致中学---借书·图书馆管理系统</title>
第7行    <link rel="stylesheet" type="text/css" href="css/style.css"/>
第8行    <script type="text/javascript" src="./js/jquery-3.4.1.min.js"></script>
第9行   </head>
第10行   <body style="">
第11行    <div id="main">
第12行        <br><br><div id="gzTitle">格致中学借书系统</div><br><br>
第13行        学生证号：<input type=text id="studID"><button class="btnStyle" id="searchStudID">查询</button>
第14行        <span id="studData"></span><br>
第15行        图书名称：<input type=text id="bookName"><button id="searchBookName" class="btnStyle">查询</button><br>
第16行        <button id="btnBorrow" class="btnStyle">借选中的图书</button>  <input type="date" id="borrowDate">
第17行        <a href="/" style="margin-left:4em">回到首页</a>
第18行        <div id="bookList"></div>
第19行    </div>
第20行    <script type="text/javascript">
第21行        $("#searchStudID").click(function(){
第22行            var studID=$.trim($("#studID").val());
第23行            if (studID.length==0)return false;/*没有填入学生证号*/
第24行            $.post("/studID",{studID:studID},function(rtn){
第25行                var rtnJSON=JSON.parse(rtn);
第26行                if (rtnJSON["state"]==0){
第27行                    $("#studData").html("学号不存在!").addClass("Caution");
第28行                }else{
第29行                    $("#studData").html("姓名:"+rtnJSON["Name"]+"  学号:"+rtnJSON["studID"]+"  性别:"+rtnJSON["Gender"]).removeClass("Caution");
第30行                }
第31行            })
第32行        });
第33行        $("#searchBookName").click(function(){
第34行            var bookName=$.trim($("#bookName").val());
第35行            if (bookName.length==0)return false;/*没有填入学生证号*/
第36行            $.post("/bookName",{bookName:bookName},function(rtn){
第37行                console.log(rtn);
第38行                var rtnJSON=JSON.parse(rtn);
第39行                if (rtnJSON["state"]==0){
第40行                    $("#bookList").html("不存在!").addClass("Caution");
第41行                }else{
第42行                    content="<table cellpadding='0' cellspacing='0' border=1 style='width:100%'>"
第43行                    content+="<tr><th style='width:5%'>序号</th><th style='width:20%'>ISBN</th><th>书名</th><th style='width:20%'>出版社</th></tr>"
第44行                    var bookList=rtnJSON["bookList"];
第45行                    for (var i=0;i<bookList.length;++i){
第46行                        content+="<tr><td>"+i.toString()+"</td><td>"+bookList[i]["ISBN"]+"</td><td>"+book
```

```
List[i]["bookName"]+"</td><td>"+bookList[i]["publisher"]+"</td></tr>"
                    }
                    content+="</table>";
                    $("#bookList").html(content).removeClass("Caution");
                }
            })
        });
        $("#bookList").on("click","tr:gt(0)",function(){
            $(this).toggleClass("lightIt");
        });
        $("#btnBorrow").click(function(){
            var selectedNode=$(".lightIt");
            if(selectedNode.length==0)return false;
            var studID=$.trim($("#studID").val());
            if(studID.length==0)return false;
            var isbnList="";
            $.each(selectedNode,function(i,item){
                isbnList+=","+$(item).children("td").eq(1).html();
            });
            isbnList=isbnList.slice(1);
            var borrowDate=$("#borrowDate").val();
            if(borrowDate.length==0)return false;
            var sendData={"studID":studID,"isbnList":isbnList,"borrowDate":borrowDate};
            $.post("/Borrow",sendData,function(rtn){
                alert(rtn);
            });
        });
    </script>
 </body>
</html>
```

例程5-21

```
<!--保存在static文件夹下的return.html-->
<!doctype html>
<html lang="zh-cn">
 <head>
  <meta charset="UTF-8">
  <title>格致中学---还书·图书馆管理系统</title>
    <link rel="stylesheet" type="text/css" href="css/style.css"/>
    <script type="text/javascript" src="./js/jquery-3.4.1.min.js"></script>
 </head>
 <body style="">
  <div id="main">
      <br><br><div id="gzTitle">格致中学还书系统</div><br><br>
      学生证号：<input type=text id="studID">
      <button class="btnStyle" id="searchStudID">查询</button>
```

```
第15行         <span id="studData"></span><br>
第16行         <button id="btnReturn" class="btnStyle">·还·<span style="font-size:14px">选中的图书</span></button>  <input type="date" id="returnDate">
第17行         <a href="/" style="margin-left:4em">回到首页</a>
第18行         <div id="bookList"></div>
第19行     </div>
第20行     <script type="text/javascript">
第21行         $("#searchStudID").click(function(){
第22行             var studID=$.trim($("#studID").val());
第23行             if (studID.length==0)return false;/*没有填入学生证号*/
第24行             $.post("/studID_Borrow",{studID:studID},function(rtn){
第25行                 var rtnJSON=JSON.parse(rtn);
第26行                 if (rtnJSON["state"]==0){
第27行                     $("#studData").html("学号不存在!").addClass("Caution");
第28行                 }else{
第29行                     $("#studData").html("姓名:"+rtnJSON["Name"]+" 学号:"+rtnJSON["studID"]+" 性别:"+rtnJSON["Gender"]).removeClass("Caution");
第30行 
第31行                     content="<table cellpadding='0' cellspacing='0' border=1 style='width:100%'>"
第32行                     content+="<tr><th style='width:9%'>借阅编号</th><th style='width:20%'>ISBN</th><th>书名</th><th style='width:20%'>出版社</th><th style='width:12%'>借阅日期</th></tr>"
第33行                     var bookList=rtnJSON["bookList"];
第34行                     for (var i=0;i<bookList.length;++i){
第35行                         content+="<tr><td>"+bookList[i]["borrowID"]+"</td><td>"+bookList[i]["ISBN"]+"</td><td>"+bookList[i]["bookName"]+"</td><td>"+bookList[i]["publisher"]+"</td><td>"+bookList[i]["borrowDate"]+"</td></tr>"
第36行                     }
第37行                     content+="</table>";
第38行                     $("#bookList").html(content).removeClass("Caution");
第39行                 }
第40行             })
第41行         });
第42行         $("#bookList").on("click","tr:gt(0)",function(){
第43行             $(this).toggleClass("lightIt");
第44行         });
第45行         $("#btnReturn").click(function(){
第46行             var selectedNode=$(".lightIt");
第47行             if(selectedNode.length==0)return false;
第48行             var studID=$.trim($("#studID").val());
第49行             if(studID.length==0)return false;
第50行             var borrowIDList="";
第51行             $.each(selectedNode,function(i,item){
第52行                 borrowIDList+=","+$(item).children("td").eq(0).html();
第53行             });
第54行             borrowIDList=borrowIDList.slice(1);
```

```
第55行            var returnDate=$("#returnDate").val();
第56行            if(returnDate.length==0)return false;
第57行            var sendData={"studID":studID,"borrowIDList":borrowIDList,"returnDate":returnDate};
第58行            $.post("/Return",sendData,function(rtn){
第59行                alert(rtn);
第60行            });
第61行        });
第62行    </script>
第63行 </body>
第64行 </html>
```

例程5-22

```
第1行 /*保存在static/css下名为：style.css*/
第2行 body{background-image:url("/static/img/bg.png");margin:0px 0px 0px 0px}
第3行 #main{border:1px solid gray;width:800px;margin:10px auto}
第4行 #gzTitle{font-size:40px;text-align:center}
第5行 #studID{font-size:24px;width:180px;line-height:32px;text-align:Center}
第6行 #bookName,#bookSearch{font-size:24px;width:300px;line-height:32px;text-align:Center}
第7行 .btnStyle{font-size:18px;margin:18px 0px 18px 0px}
第8行 #bookList{width:100%;}
第9行 #Fx{text-align:center;font-size:32px;}
第10行 .Caution{color:red}
第11行 .lightIt{background-color:red}
```

第六节　MySQL

第七节　小结