17 | 如何使用Python操作MySQL?

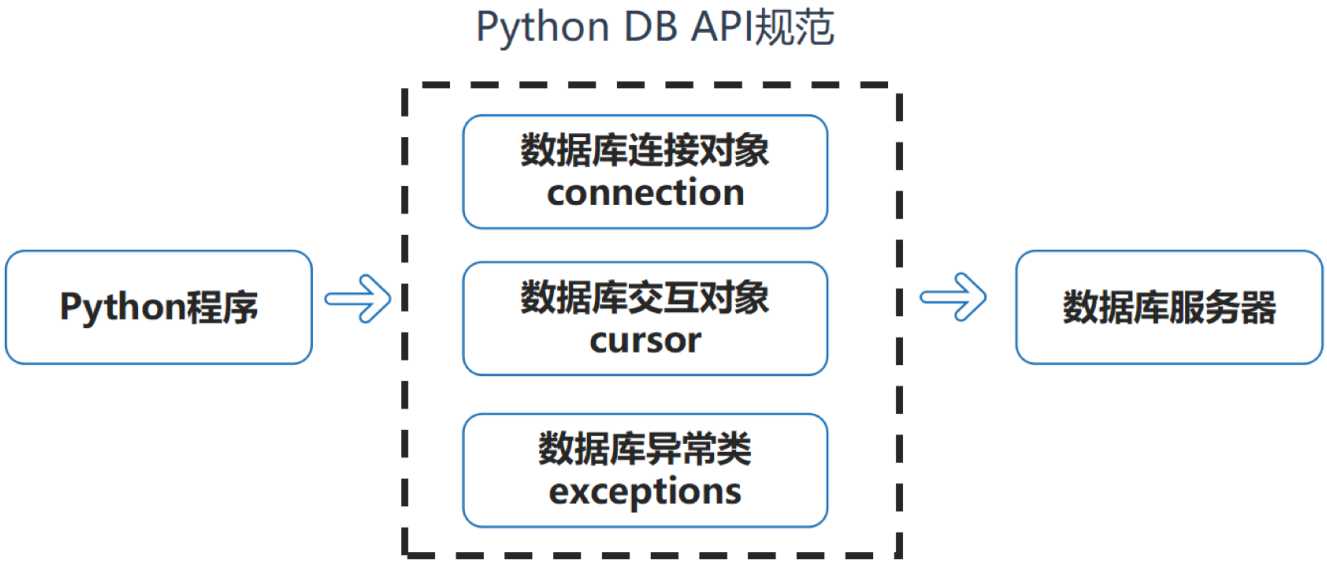
我们之前都是直接在 DBMS 里面进行 SQL 的操作，实际上我们还可以通过后端语言对 DBMS 进行访问以及进行相应的操作，这样更具有灵活性，可以实现一些较为复杂的操作。作为一个后端开发人员，掌握一些 SQL 技术是必须的；作为一个数据库管理人员，了解后端语言如何开发和管理数据库也是很有必要的。

今天我以 Python 为例，讲解下如何对 MySQL 数据库进行操作。你需要掌握以下几个方面的内容：

1. Python 的 DB API 规范是什么，遵守这个规范有什么用?
2. 基于 DB API，MySQL 官方提供了驱动器 mysql-connector，如何使用它来完成对数据库管理系统的操作?
3. CRUD 是最常见的数据库的操作，分别对应数据的增加、读取、修改和删除。在掌握了 mysql-connector 的使用方法之后，如何完成对数据表的 CRUD 操作?

Python DB API 规范

Python 可以支持非常多的数据库管理系统，比如 MySQL、Oracle、SQL Server 和 PostgreSQL 等。为了实现对这些 DBMS 的统一访问，Python 需要遵守一个规范，这就是 DB API 规范。我在下图中列出了 DB API 规范的作用，这个规范给我们提供了数据库对象连接、对象交互和异常处理的方式，为各种 DBMS 提供了统一的访问接口。这样做的好处就是如果项目需要切换数据库，Python 层的代码移植会比较简单。



我们在使用 Python 对 DBMS 进行操作的时候，需要经过下面的几个步骤：

1. 引入 API 模块；
2. 与数据库建立连接；
3. 执行 SQL 语句；
4. 关闭数据库连接。

如何使用 mysql-connector

使用 Python 对数据库进行访问需要基于 DB API 规范，这里有不少库供我们选择，比如 MySQLdb、mysqlclient、PyMySQL、peewee 和 SQLAlchemy 等。今天我讲解的是 mysql-connector，它是 MySQL 官方提供的驱动器，用来给后端语言，比如 Python 提供连接。

下面我们看下如何用 Python 使用 mysql-connector，以完成数据库的连接和使用。

首先安装 mysql-connector。在使用前，你需要先使用下面这句命令进行安装：

复制代码

```
pip install mysql-connector
```

在安装之后，你可以创建数据库连接，然后查看下数据库的版本号，来验证下数据库是否连接成功。代码如下：

复制代码

```
# -*- coding: UTF-8 -*-
import mysql.connector
# 打开数据库连接
db = mysql.connector.connect(
        host="localhost",
        user="root",
        passwd="XXX", # 写上你的数据库密码
        database='wucai',
        auth_plugin='mysql_native_password'
)
# 获取操作游标
cursor = db.cursor()
# 执行 SQL 语句
```

```
cursor.execute("SELECT VERSION()")
# 获取一条数据
data = cursor.fetchone()
print("MySQL 版本: %s " % data)
# 关闭游标 & 数据库连接
cursor.close()
db.close()
```

运行结果：

复制代码

```
MySQL 版本: 8.0.13
```

上面这段代码中有两个重要的对象你需要了解下，分别是 Connection 和 Cursor。

Connection 就是对数据库的当前连接进行管理，我们可以通过它来进行以下操作：

1. 通过指定 host、user、passwd 和 port 等参数来创建数据库连接，这些参数分别对应着数据库 IP 地址、用户名、密码和端口号；
2. 使用 db.close() 关闭数据库连接；
3. 使用 db.cursor() 创建游标，操作数据库中的数据；
4. 使用 db.begin() 开启事务；
5. 使用 db.commit() 和 db.rollback()，对事务进行提交以及回滚。

当我们通过`cursor = db.cursor()`创建游标后，就可以通过面向过程的编程方式对数据库中的数据进行操作：

1. 使用`cursor.execute(query_sql)`，执行数据库查询；
2. 使用`cursor.fetchone()`，读取数据集中的一条数据；
3. 使用`cursor.fetchall()`，取出数据集中的所有行，返回一个元组 tuples 类型；
4. 使用`cursor.fetchmany(n)`，取出数据集中的多条数据，同样返回一个元组 tuples；
5. 使用`cursor.rowcount`，返回查询结果集中的行数。如果没有查询到数据或者还没有查询，则结果为 -1，否则会返回查询得到的数据行数；
6. 使用`cursor.close()`，关闭游标。

对数据表进行增删改查

了解了 Connection 和 Cursor 的使用方式之后，我们来看下如何来对 heros 数据表进行 CRUD 的操作，即增加、读取、更新和删除。

首先是增加数据。

假设我们想在 player 表中增加一名新球员，姓名为“约翰·科林斯”，球队 ID 为 1003（即亚特兰大老鹰），身高为 2.08m。代码如下：

复制代码

```
# 插入新球员
sql = "INSERT INTO player (team_id, player_name, height) VALUES (%s, %s, %s)"
val = (1003, " 约翰 - 科林斯 ", 2.08)
cursor.execute(sql, val)
db.commit()
print(cursor.rowcount, " 记录插入成功。")
```

我们使用 cursor.execute 来执行相应的 SQL 语句，val 为 SQL 语句中的参数，SQL 执行后使用 db.commit() 进行提交。需要说明的是，我们在使用 SQL 语句的时候，可以向 SQL 语句传递参数，这时 SQL 语句里要统一用（%s）进行占位，否则就会报错。不论插入的数值为整数类型，还是浮点类型，都需要统一用（%s）进行占位。

另外在用游标进行 SQL 操作之后，还需要使用 db.commit() 进行提交，否则数据不会被插入。

然后是读取数据。我们来看下数据是否被插入成功，这里我们查询下身高大于等于 2.08m 的球员都有哪些，代码如下：

复制代码

```
# 查询身高大于等于 2.08 的球员
sql = 'SELECT player_id, player_name, height FROM player WHERE height>=2.08'
cursor.execute(sql)
data = cursor.fetchall()
for each_player in data:
  print(each_player)
```

运行结果：

复制代码

```
(10003, '安德烈 - 德拉蒙德', 2.11)
(10004, '索恩 - 马克', 2.16)
(10009, '扎扎 - 帕楚里亚', 2.11)
(10010, '乔恩 - 洛伊尔', 2.08)
(10011, '布雷克 - 格里芬', 2.08)
(10015, '亨利 - 埃伦森', 2.11)
(10023, '多曼塔斯 - 萨博尼斯', 2.11)
(10024, '迈尔斯 - 特纳', 2.11)
(10032, 'TJ- 利夫', 2.08)
(10033, '凯尔 - 奥奎因', 2.08)
(10037, '伊凯·阿尼博古', 2.08)
(10038, '约翰 - 科林斯', 2.08)
```

你能看到球员约翰·科林斯被正确插入。

那么如何修改数据呢？

假如我想修改刚才插入的球员约翰·科林斯的身高，将身高修改成 2.09，代码如下：

复制代码

```
# 修改球员约翰 - 科林斯
sql = 'UPDATE player SET height = %s WHERE player_name = %s'
val = (2.09, " 约翰 - 科林斯 ")
cursor.execute(sql, val)
db.commit()
print(cursor.rowcount, " 记录被修改。")
```

最后我们看下如何删除约翰·科林斯这个球员的数据，代码如下：

复制代码

```
sql = 'DELETE FROM player WHERE player_name = %s'
val = (" 约翰 - 科林斯 ",)
cursor.execute(sql, val)
db.commit()
print(cursor.rowcount, " 记录删除成功。")
```

最后都执行完了，我们来关闭游标和数据库的连接，使用以下代码即可：

复制代码

```
cursor.close()
db.close()
```

针对上面的操作过程，你可以模拟下数据的 CRUD 操作，但有几点你需要注意。

1. 打开数据库连接以后，如果不再使用，则需要关闭数据库连接，以免造成资源浪费。
2. 在对数据进行增加、删除和修改的时候，可能会出现异常，这时就需要用`try...except`捕获异常信息。比如针对插入球员约翰·科林斯这个操作，你可以写成下面这样：

复制代码

```
import traceback
try:
  sql = "INSERT INTO player (team_id, player_name, height) VALUES (%s, %s, %s)"
  val = (1003, " 约翰 - 科林斯 ", 2.08)
  cursor.execute(sql, val)
  db.commit()
  print(cursor.rowcount, " 记录插入成功。")
except Exception as e:
  # 打印异常信息
  traceback.print_exc()
  # 回滚
  db.rollback()
finally:
  # 关闭数据库连接
  db.close()
```

运行结果告诉我们记录插入成功。

3. 如果你在使用 mysql-connector 连接的时候，系统报的错误为`authentication plugin caching_sha2`，这时你需要下载最新的版本更新来解决，点击这里进行更新。

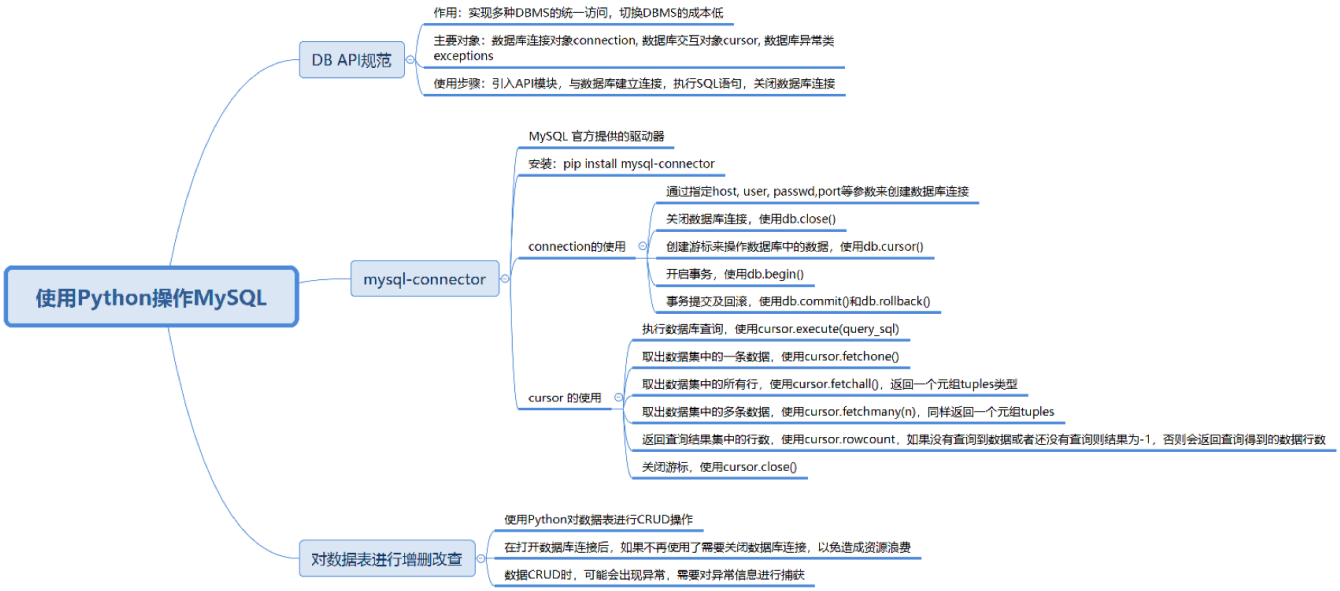
总结

我今天讲解了如何使用 Python 来操作 MySQL，这里我们使用的是官方提供的 mysql-connector，当然除了它之外，还有很多库可以进行选择。

在使用基于 DB API 规范的协议时，重点需要掌握 Connection 和 Cursor 这两个对象，Connection 就是对数据库的连接进行管理，而 Cursor 是对数据库的游标进行管理，通过它们，我们可以执行具体的 SQL 语句，以及处理复杂的数据。

用 Python 操作 MySQL，还有很多种姿势，mysql-connector 只是其中一种，实际上还有另外一种方式，就是采用 ORM 框架。ORM 的英文是 Object Relational Mapping，也就是采用对象关系映射的模式，使用这种模式可以将数据库中各种数据表之间的关系映射到程序中的对象。这种模式可以屏蔽底层的数据库的细节，不需要我们与复杂的 SQL 语句打交道，直接采用操作对象的形式操作就可以。

不过如果应用数据实体少，其实没有必要使用 ORM 框架，针对少量对象的管理，自己实现起来也很简单，比如本篇文章中我讲到的采用官方提供的 mysql-connector 驱动的方式来实现 CRUD。引入一个框架的学习成本很高，代码膨胀也很厉害，所以如果是相对简单的操作，完全可以自己动手来实现。



使用 Python 对数据库进行操作，关键在于实战，所以这里我出一个练习题。请你使用 Python 对 heros 表中最大生命值大于 6000 的英雄进行查询，并且输出相应的属性值。