## 创建数据库连接

<https://www.runoob.com/python3/python-mysql-connector.html>

可以使用以下代码来连接数据库：

## demo\_mysql\_test.py:

import mysql.connector

mydb = mysql.connector.connect( host="localhost", # 数据库主机地址 user="yourusername", # 数据库用户名 passwd="yourpassword" # 数据库密码 )

print(mydb)

### **创建数据库**

创建数据库使用 "CREATE DATABASE" 语句，以下创建一个名为 runoob\_db 的数据库：

## demo\_mysql\_test.py:

import mysql.connector mydb = mysql.connector.connect( host="localhost", user="root", passwd="123456" ) mycursor = mydb.cursor() mycursor.execute("CREATE DATABASE runoob\_db")

创建数据库前我们也可以使用 "SHOW DATABASES" 语句来查看数据库是否存在：

## demo\_mysql\_test.py:

输出所有数据库列表：

import mysql.connector mydb = mysql.connector.connect( host="localhost", user="root", passwd="123456" ) mycursor = mydb.cursor() mycursor.execute("SHOW DATABASES") for x in mycursor: print(x)

或者我们可以直接连接数据库，如果数据库不存在，会输出错误信息：

## demo\_mysql\_test.py:

import mysql.connector mydb = mysql.connector.connect( host="localhost", user="root", passwd="123456", database="runoob\_db" )

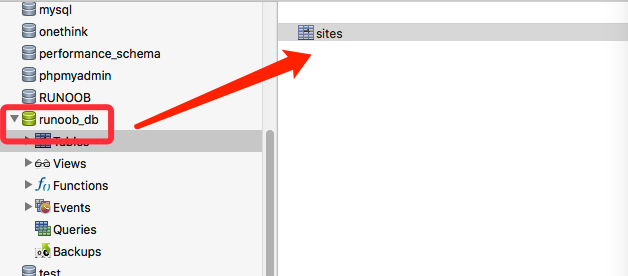
## 创建数据表

创建数据表使用 **"CREATE TABLE"** 语句，创建数据表前，需要确保数据库已存在，以下创建一个名为 **sites** 的数据表：

## demo\_mysql\_test.py:

import mysql.connector mydb = mysql.connector.connect( host="localhost", user="root", passwd="123456", database="runoob\_db" ) mycursor = mydb.cursor() mycursor.execute("CREATE TABLE sites (name VARCHAR(255), url VARCHAR(255))")

执行成功后，我们可以看到数据库创建的数据表 sites，字段为 name 和 url。



**我们也可以使用 "SHOW TABLES" 语句来查看数据表是否已存在：**

## demo\_mysql\_test.py:

import mysql.connector mydb = mysql.connector.connect( host="localhost", user="root", passwd="123456", database="runoob\_db" ) mycursor = mydb.cursor() mycursor.execute("SHOW TABLES") for x in mycursor: print(x)

### **主键设置**

创建表的时候我们一般都会设置一个主键（PRIMARY KEY），我们可以使用 **"INT AUTO\_INCREMENT PRIMARY KEY"** 语句来创建一个主键，主键起始值为 1，逐步递增。

如果我们的表已经创建，我们需要使用 **ALTER TABLE** 来给表添加主键：

## demo\_mysql\_test.py:

给 sites 表添加主键。

import mysql.connector mydb = mysql.connector.connect( host="localhost", user="root", passwd="123456", database="runoob\_db" ) mycursor = mydb.cursor() mycursor.execute("ALTER TABLE sites ADD COLUMN id INT AUTO\_INCREMENT PRIMARY KEY")

如果你还未创建 sites 表，可以直接使用以下代码创建。

## demo\_mysql\_test.py:

给表创建主键。

import mysql.connector mydb = mysql.connector.connect( host="localhost", user="root", passwd="123456", database="runoob\_db" ) mycursor = mydb.cursor() mycursor.execute("CREATE TABLE sites (id INT AUTO\_INCREMENT PRIMARY KEY, name VARCHAR(255), url VARCHAR(255))")

## 插入数据

插入数据使用 **"INSERT INTO"** 语句：

## demo\_mysql\_test.py:

向 sites 表插入一条记录。

import mysql.connector mydb = mysql.connector.connect( host="localhost", user="root", passwd="123456", database="runoob\_db" ) mycursor = mydb.cursor() sql = "INSERT INTO sites (name, url) VALUES (%s, %s)" val = ("RUNOOB", "https://www.runoob.com") mycursor.execute(sql, val) mydb.commit() # 数据表内容有更新，必须使用到该语句 print(mycursor.rowcount, "记录插入成功。")

执行代码，输出结果为：

1 记录插入成功

### **批量插入**

批量插入使用 **executemany()** 方法，该方法的第二个参数是一个元组列表，包含了我们要插入的数据：

## demo\_mysql\_test.py:

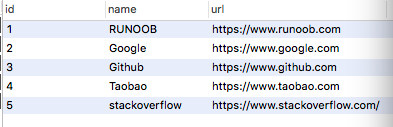
向 sites 表插入多条记录。

import mysql.connector mydb = mysql.connector.connect( host="localhost", user="root", passwd="123456", database="runoob\_db" ) mycursor = mydb.cursor() sql = "INSERT INTO sites (name, url) VALUES (%s, %s)" val = [ ('Google', 'https://www.google.com'), ('Github', 'https://www.github.com'), ('Taobao', 'https://www.taobao.com'), ('stackoverflow', 'https://www.stackoverflow.com/') ] mycursor.executemany(sql, val) mydb.commit() # 数据表内容有更新，必须使用到该语句 print(mycursor.rowcount, "记录插入成功。")

执行代码，输出结果为：

4 记录插入成功。

执行以上代码后，我们可以看看数据表的记录：



如果我们想在数据记录插入后，获取该记录的 ID ，可以使用以下代码：

## demo\_mysql\_test.py:

import mysql.connector mydb = mysql.connector.connect( host="localhost", user="root", passwd="123456", database="runoob\_db" ) mycursor = mydb.cursor() sql = "INSERT INTO sites (name, url) VALUES (%s, %s)" val = ("Zhihu", "https://www.zhihu.com") mycursor.execute(sql, val) mydb.commit() print("1 条记录已插入, ID:", mycursor.lastrowid)

执行代码，输出结果为：

1 条记录已插入, ID: 6

## 查询数据

查询数据使用 **SELECT** 语句：

## demo\_mysql\_test.py:

import mysql.connector mydb = mysql.connector.connect( host="localhost", user="root", passwd="123456", database="runoob\_db" ) mycursor = mydb.cursor() mycursor.execute("SELECT \* FROM sites") myresult = mycursor.fetchall() # fetchall() 获取所有记录 for x in myresult: print(x)

执行代码，输出结果为：

(1, 'RUNOOB', 'https://www.runoob.com')(2, 'Google', 'https://www.google.com')(3, 'Github', 'https://www.github.com')(4, 'Taobao', 'https://www.taobao.com')(5, 'stackoverflow', 'https://www.stackoverflow.com/')(6, 'Zhihu', 'https://www.zhihu.com')

也可以读取指定的字段数据：

## demo\_mysql\_test.py:

import mysql.connector mydb = mysql.connector.connect( host="localhost", user="root", passwd="123456", database="runoob\_db" ) mycursor = mydb.cursor() mycursor.execute("SELECT name, url FROM sites") myresult = mycursor.fetchall() for x in myresult: print(x)

执行代码，输出结果为：

('RUNOOB', 'https://www.runoob.com')('Google', 'https://www.google.com')('Github', 'https://www.github.com')('Taobao', 'https://www.taobao.com')('stackoverflow', 'https://www.stackoverflow.com/')('Zhihu', 'https://www.zhihu.com')

如果我们只想读取一条数据，可以使用 **fetchone()** 方法：

## demo\_mysql\_test.py:

import mysql.connector mydb = mysql.connector.connect( host="localhost", user="root", passwd="123456", database="runoob\_db" ) mycursor = mydb.cursor() mycursor.execute("SELECT \* FROM sites") myresult = mycursor.fetchone() print(myresult)

执行代码，输出结果为：

(1, 'RUNOOB', 'https://www.runoob.com')

### **where 条件语句**

如果我们要读取指定条件的数据，可以使用 **where** 语句：

## demo\_mysql\_test.py

读取 name 字段为 RUNOOB 的记录：

import mysql.connector mydb = mysql.connector.connect( host="localhost", user="root", passwd="123456", database="runoob\_db" ) mycursor = mydb.cursor() sql = "SELECT \* FROM sites WHERE name ='RUNOOB'" mycursor.execute(sql) myresult = mycursor.fetchall() for x in myresult: print(x)

执行代码，输出结果为：

(1, 'RUNOOB', 'https://www.runoob.com')

也可以使用通配符 **%**：

## demo\_mysql\_test.py

import mysql.connector mydb = mysql.connector.connect( host="localhost", user="root", passwd="123456", database="runoob\_db" ) mycursor = mydb.cursor() sql = "SELECT \* FROM sites WHERE url LIKE '%oo%'" mycursor.execute(sql) myresult = mycursor.fetchall() for x in myresult: print(x)

执行代码，输出结果为：

(1, 'RUNOOB', 'https://www.runoob.com')(2, 'Google', 'https://www.google.com')

为了防止数据库查询发生 SQL 注入的攻击，我们可以使用 **%s** 占位符来转义查询的条件：

## demo\_mysql\_test.py

import mysql.connector mydb = mysql.connector.connect( host="localhost", user="root", passwd="123456", database="runoob\_db" ) mycursor = mydb.cursor() sql = "SELECT \* FROM sites WHERE name = %s" na = ("RUNOOB", ) mycursor.execute(sql, na) myresult = mycursor.fetchall() for x in myresult: print(x)

### **排序**

查询结果排序可以使用 **ORDER BY** 语句，默认的排序方式为升序，关键字为 **ASC**，如果要设置降序排序，可以设置关键字 **DESC**。

## demo\_mysql\_test.py

按 name 字段字母的升序排序：

import mysql.connector mydb = mysql.connector.connect( host="localhost", user="root", passwd="123456", database="runoob\_db" ) mycursor = mydb.cursor() sql = "SELECT \* FROM sites ORDER BY name" mycursor.execute(sql) myresult = mycursor.fetchall() for x in myresult: print(x)

执行代码，输出结果为：

(3, 'Github', 'https://www.github.com')(2, 'Google', 'https://www.google.com')(1, 'RUNOOB', 'https://www.runoob.com')(5, 'stackoverflow', 'https://www.stackoverflow.com/')(4, 'Taobao', 'https://www.taobao.com')(6, 'Zhihu', 'https://www.zhihu.com')

降序排序实例：

## demo\_mysql\_test.py

按 name 字段字母的降序排序：

import mysql.connector mydb = mysql.connector.connect( host="localhost", user="root", passwd="123456", database="runoob\_db" ) mycursor = mydb.cursor() sql = "SELECT \* FROM sites ORDER BY name DESC" mycursor.execute(sql) myresult = mycursor.fetchall() for x in myresult: print(x)

执行代码，输出结果为：

(6, 'Zhihu', 'https://www.zhihu.com')(4, 'Taobao', 'https://www.taobao.com')(5, 'stackoverflow', 'https://www.stackoverflow.com/')(1, 'RUNOOB', 'https://www.runoob.com')(2, 'Google', 'https://www.google.com')(3, 'Github', 'https://www.github.com')

### **Limit**

如果我们要设置查询的数据量，可以通过 **"LIMIT"** 语句来指定

## demo\_mysql\_test.py

读取前 3 条记录：

import mysql.connector mydb = mysql.connector.connect( host="localhost", user="root", passwd="123456", database="runoob\_db" ) mycursor = mydb.cursor() mycursor.execute("SELECT \* FROM sites LIMIT 3") myresult = mycursor.fetchall() for x in myresult: print(x)

执行代码，输出结果为：

(1, 'RUNOOB', 'https://www.runoob.com')(2, 'Google', 'https://www.google.com')(3, 'Github', 'https://www.github.com')

也可以指定起始位置，使用的关键字是 **OFFSET**：

## demo\_mysql\_test.py

从第二条开始读取前 3 条记录：

import mysql.connector mydb = mysql.connector.connect( host="localhost", user="root", passwd="123456", database="runoob\_db" ) mycursor = mydb.cursor() mycursor.execute("SELECT \* FROM sites LIMIT 3 OFFSET 1") # 0 为 第一条，1 为第二条，以此类推 myresult = mycursor.fetchall() for x in myresult: print(x)

执行代码，输出结果为：

(2, 'Google', 'https://www.google.com')(3, 'Github', 'https://www.github.com')(4, 'Taobao', 'https://www.taobao.com')

## 删除记录

删除记录使用 **"DELETE FROM"** 语句：

## demo\_mysql\_test.py

删除 name 为 stackoverflow 的记录：

import mysql.connector mydb = mysql.connector.connect( host="localhost", user="root", passwd="123456", database="runoob\_db" ) mycursor = mydb.cursor() sql = "DELETE FROM sites WHERE name = 'stackoverflow'" mycursor.execute(sql) mydb.commit() print(mycursor.rowcount, " 条记录删除")

执行代码，输出结果为：

1 条记录删除

**注意：**要慎重使用删除语句，删除语句要确保指定了 WHERE 条件语句，否则会导致整表数据被删除。

为了防止数据库查询发生 SQL 注入的攻击，我们可以使用 **%s** 占位符来转义删除语句的条件：

## demo\_mysql\_test.py

import mysql.connector mydb = mysql.connector.connect( host="localhost", user="root", passwd="123456", database="runoob\_db" ) mycursor = mydb.cursor() sql = "DELETE FROM sites WHERE name = %s" na = ("stackoverflow", ) mycursor.execute(sql, na) mydb.commit() print(mycursor.rowcount, " 条记录删除")

执行代码，输出结果为：

1 条记录删除

## 更新表数据

数据表更新使用 **"UPDATE"** 语句：

## demo\_mysql\_test.py

将 name 为 Zhihu 的字段数据改为 ZH：

import mysql.connector mydb = mysql.connector.connect( host="localhost", user="root", passwd="123456", database="runoob\_db" ) mycursor = mydb.cursor() sql = "UPDATE sites SET name = 'ZH' WHERE name = 'Zhihu'" mycursor.execute(sql) mydb.commit() print(mycursor.rowcount, " 条记录被修改")

执行代码，输出结果为：

1 条记录被修改

**注意：**UPDATE 语句要确保指定了 WHERE 条件语句，否则会导致整表数据被更新。

为了防止数据库查询发生 SQL 注入的攻击，我们可以使用 %s 占位符来转义更新语句的条件：

## demo\_mysql\_test.py

import mysql.connector mydb = mysql.connector.connect( host="localhost", user="root", passwd="123456", database="runoob\_db" ) mycursor = mydb.cursor() sql = "UPDATE sites SET name = %s WHERE name = %s" val = ("Zhihu", "ZH") mycursor.execute(sql, val) mydb.commit() print(mycursor.rowcount, " 条记录被修改")

执行代码，输出结果为：

1 条记录被修改

## 删除表

删除表使用 **"DROP TABLE"** 语句， **IF EXISTS** 关键字是用于判断表是否存在，只有在存在的情况才删除：

## demo\_mysql\_test.py

import mysql.connector mydb = mysql.connector.connect( host="localhost", user="root", passwd="123456", database="runoob\_db" ) mycursor = mydb.cursor() sql = "DROP TABLE IF EXISTS sites" # 删除数据表 sites mycursor.execute(sql)

### **实例：**

以下实例链接 Mysql 的 TESTDB 数据库：

## 实例(Python 3.0+)

#!/usr/bin/python3 import pymysql # 打开数据库连接 db = pymysql.connect("localhost","testuser","test123","TESTDB" ) # 使用 cursor() 方法创建一个游标对象 cursor cursor = db.cursor() # 使用 execute() 方法执行 SQL 查询 cursor.execute("SELECT VERSION()") # 使用 fetchone() 方法获取单条数据. data = cursor.fetchone() print ("Database version : %s " % data) # 关闭数据库连接 db.close()

执行以上脚本输出结果如下：

Database version : 5.5.20-log

## 创建数据库表

如果数据库连接存在我们可以使用execute()方法来为数据库创建表，如下所示创建表EMPLOYEE：

## 实例(Python 3.0+)

#!/usr/bin/python3 import pymysql # 打开数据库连接 db = pymysql.connect("localhost","testuser","test123","TESTDB" ) # 使用 cursor() 方法创建一个游标对象 cursor cursor = db.cursor() # 使用 execute() 方法执行 SQL，如果表存在则删除 cursor.execute("DROP TABLE IF EXISTS EMPLOYEE") # 使用预处理语句创建表 sql = """CREATE TABLE EMPLOYEE ( FIRST\_NAME CHAR(20) NOT NULL, LAST\_NAME CHAR(20), AGE INT, SEX CHAR(1), INCOME FLOAT )""" cursor.execute(sql) # 关闭数据库连接 db.close()

## 数据库插入操作

以下实例使用执行 SQL INSERT 语句向表 EMPLOYEE 插入记录：

## 实例(Python 3.0+)

#!/usr/bin/python3 import pymysql # 打开数据库连接 db = pymysql.connect("localhost","testuser","test123","TESTDB" ) # 使用cursor()方法获取操作游标 cursor = db.cursor() # SQL 插入语句 sql = """INSERT INTO EMPLOYEE(FIRST\_NAME, LAST\_NAME, AGE, SEX, INCOME) VALUES ('Mac', 'Mohan', 20, 'M', 2000)""" try: # 执行sql语句 cursor.execute(sql) # 提交到数据库执行 db.commit() except: # 如果发生错误则回滚 db.rollback() # 关闭数据库连接 db.close()

以上例子也可以写成如下形式：

## 实例(Python 3.0+)

#!/usr/bin/python3 import pymysql # 打开数据库连接 db = pymysql.connect("localhost","testuser","test123","TESTDB" ) # 使用cursor()方法获取操作游标 cursor = db.cursor() # SQL 插入语句 sql = "INSERT INTO EMPLOYEE(FIRST\_NAME, \ LAST\_NAME, AGE, SEX, INCOME) \ VALUES ('%s', '%s', %s, '%s', %s)" % \ ('Mac', 'Mohan', 20, 'M', 2000) try: # 执行sql语句 cursor.execute(sql) # 执行sql语句 db.commit() except: # 发生错误时回滚 db.rollback() # 关闭数据库连接 db.close()

以下代码使用变量向SQL语句中传递参数:

..................................

user\_id = "test123"

password = "password"

con.execute('insert into Login values( %s, %s)' % \

(user\_id, password))..................................

## 数据库查询操作

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。

* **fetchone():** 该方法获取下一个查询结果集。结果集是一个对象
* **fetchall():**接收全部的返回结果行.
* **rowcount:** 这是一个只读属性，并返回执行execute()方法后影响的行数。

### **实例：**

查询EMPLOYEE表中salary（工资）字段大于1000的所有数据：

## 实例(Python 3.0+)

#!/usr/bin/python3 import pymysql # 打开数据库连接 db = pymysql.connect("localhost","testuser","test123","TESTDB" ) # 使用cursor()方法获取操作游标 cursor = db.cursor() # SQL 查询语句 sql = "SELECT \* FROM EMPLOYEE \ WHERE INCOME > %s" % (1000) try: # 执行SQL语句 cursor.execute(sql) # 获取所有记录列表 results = cursor.fetchall() for row in results: fname = row[0] lname = row[1] age = row[2] sex = row[3] income = row[4] # 打印结果 print ("fname=%s,lname=%s,age=%s,sex=%s,income=%s" % \ (fname, lname, age, sex, income )) except: print ("Error: unable to fetch data") # 关闭数据库连接 db.close()

以上脚本执行结果如下：

fname=Mac, lname=Mohan, age=20, sex=M, income=2000

## 数据库更新操作

更新操作用于更新数据表的的数据，以下实例将 TESTDB 表中 SEX 为 'M' 的 AGE 字段递增 1：

## 实例(Python 3.0+)

#!/usr/bin/python3 import pymysql # 打开数据库连接 db = pymysql.connect("localhost","testuser","test123","TESTDB" ) # 使用cursor()方法获取操作游标 cursor = db.cursor() # SQL 更新语句 sql = "UPDATE EMPLOYEE SET AGE = AGE + 1 WHERE SEX = '%c'" % ('M') try: # 执行SQL语句 cursor.execute(sql) # 提交到数据库执行 db.commit() except: # 发生错误时回滚 db.rollback() # 关闭数据库连接 db.close()

## 删除操作

删除操作用于删除数据表中的数据，以下实例演示了删除数据表 EMPLOYEE 中 AGE 大于 20 的所有数据：

## 实例(Python 3.0+)

#!/usr/bin/python3 import pymysql # 打开数据库连接 db = pymysql.connect("localhost","testuser","test123","TESTDB" ) # 使用cursor()方法获取操作游标 cursor = db.cursor() # SQL 删除语句 sql = "DELETE FROM EMPLOYEE WHERE AGE > %s" % (20) try: # 执行SQL语句 cursor.execute(sql) # 提交修改 db.commit() except: # 发生错误时回滚 db.rollback() # 关闭连接 db.close()

## 执行事务

事务机制可以确保数据一致性。

事务应该具有4个属性：原子性、一致性、隔离性、持久性。这四个属性通常称为ACID特性。

* 原子性（atomicity）。一个事务是一个不可分割的工作单位，事务中包括的诸操作要么都做，要么都不做。
* 一致性（consistency）。事务必须是使数据库从一个一致性状态变到另一个一致性状态。一致性与原子性是密切相关的。
* 隔离性（isolation）。一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。
* 持久性（durability）。持续性也称永久性（permanence），指一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。接下来的其他操作或故障不应该对其有任何影响。

Python DB API 2.0 的事务提供了两个方法 commit 或 rollback。

### **实例**

## 实例(Python 3.0+)

# SQL删除记录语句 sql = "DELETE FROM EMPLOYEE WHERE AGE > %s" % (20) try: # 执行SQL语句 cursor.execute(sql) # 向数据库提交 db.commit() except: # 发生错误时回滚 db.rollback()

对于支持事务的数据库， 在Python数据库编程中，当游标建立之时，就自动开始了一个隐形的数据库事务。

commit()方法游标的所有更新操作，rollback（）方法回滚当前游标的所有操作。每一个方法都开始了一个新的事务。

<https://www.runoob.com/python3/python3-mysql.html>