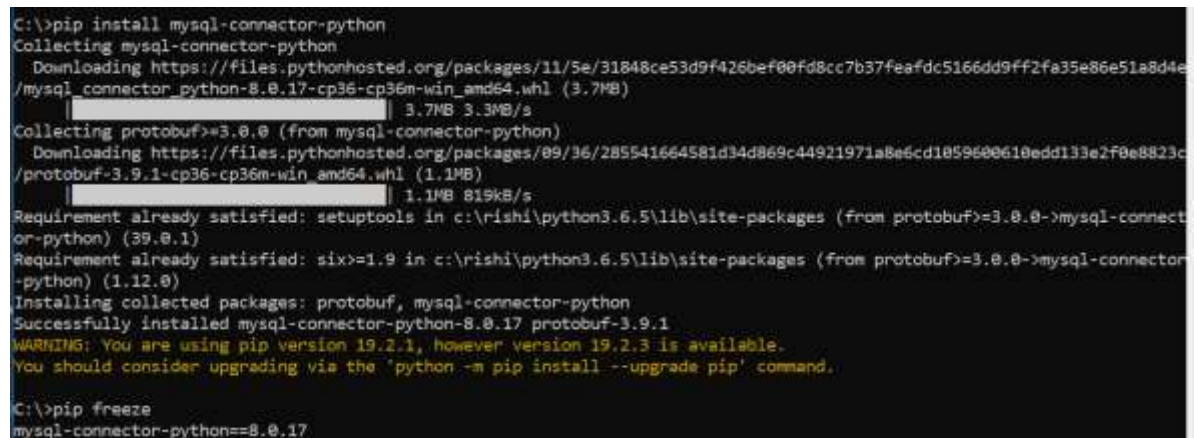


Install MySQL Driver

Python needs a MySQL driver to access the MySQL database. In this tutorial we will use the driver "MySQL Connector". We recommend that you use PIP to install "MySQL Connector". PIP is most likely already installed in your Python environment.

Navigate your command line to the location of PIP, and type the following:

```
python -m pip install mysql-connector-python
```



```
C:\>pip install mysql-connector-python
Collecting mysql-connector-python
  Downloading https://files.pythonhosted.org/packages/11/5e/31848ce53d9f426bef00fd8cc7b37feafdc5166dd9ff2fa35e86e51a8d4e
/mysql_connector_python-8.0.17-cp36-cp36m-win_amd64.whl (3.7MB)
    |#####| 3.7MB 3.3MB/s
Collecting protobuf>=3.0.0 (from mysql-connector-python)
  Downloading https://files.pythonhosted.org/packages/89/36/285541664581d34d869c44921971a8e6cd1059600610edd133e2f8e8823c
/protobuf-3.9.1-cp36-cp36m-win_amd64.whl (1.1MB)
    |#####| 1.1MB 819kB/s
Requirement already satisfied: setuptools in c:\rishi\python3.6.5\lib\site-packages (from protobuf>=3.0.0->mysql-connect
or-python) (39.0.1)
Requirement already satisfied: six>=1.9 in c:\rishi\python3.6.5\lib\site-packages (from protobuf>=3.0.0->mysql-connector
-python) (1.12.0)
Installing collected packages: protobuf, mysql-connector-python
Successfully installed mysql-connector-python-8.0.17 protobuf-3.9.1
WARNING: You are using pip version 19.2.1, however version 19.2.3 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\>pip freeze
mysql-connector-python==8.0.17
```

Now you have downloaded and installed a MySQL driver.

Test MySQL Connector

To test if the installation was successful, or if you already have "MySQL Connector" installed, create a Python page with the following content:

```
import mysql.connector
```

If the above code was executed with no errors, "MySQL Connector" is installed and ready to be used.

Create Connection

Start by creating a connection to the database.

Use the username and password from your MySQL database:

```
import mysql.connector
```

```
mydb = mysql.connector.connect(
    host="localhost",
    user="yourusername",
```

```
    passwd="yourpassword"  
)
```

```
print(mydb)
```

Now you can start querying the database using SQL statements.

Creating a Database

To create a database in MySQL, use the "CREATE DATABASE" statement:

```
import mysql.connector
```

```
mydb = mysql.connector.connect(  
    host="localhost",  
    user="yourusername",  
    passwd="yourpassword"  
)
```

```
mycursor = mydb.cursor()
```

```
mycursor.execute("CREATE DATABASE mydatabase")
```

If the above code was executed with no errors, you have successfully created a database.

Check if Database Exists

You can check if a database exist by listing all databases in your system by using the "SHOW DATABASES" statement:

```
mycursor = mydb.cursor()
```

```
mycursor.execute("SHOW DATABASES")
```

```
for x in mycursor:  
    print(x)
```

Or you can try to access the database when making the connection:

```
import mysql.connector
```

```
mydb = mysql.connector.connect(  
    host="localhost",  
    user="yourusername",  
    passwd="yourpassword",  
    database="mydatabase"  
)
```

If the database does not exist, you will get an error.

Creating a Table

To create a table in MySQL, use the "CREATE TABLE" statement.

Make sure you define the name of the database when you create the connection

```
mycursor = mydb.cursor()
```

```
mycursor.execute("CREATE TABLE customers (name VARCHAR(255), address  
VARCHAR(255))")
```

If the above code was executed with no errors, you have now successfully created a table.

Check if Table Exists

You can check if a table exist by listing all tables in your database with the "SHOW TABLES" statement:

```
mycursor.execute("SHOW TABLES")
```

```
for x in mycursor:  
    print(x)
```

Insert Into Table

To fill a table in MySQL, use the "INSERT INTO" statement.

```
mycursor = mydb.cursor()
```

```
sql = "INSERT INTO mydatabase.customers (name, address) VALUES (%s, %s)"  
val = ("John", "Highway 21")  
mycursor.execute(sql, val)
```

```
mydb.commit()
```

```
print(mycursor.rowcount, "record inserted.")
```

Important!: Notice the statement: `mydb.commit()`. It is required to make the changes, otherwise no changes are made to the table.

Insert Multiple Rows

To insert multiple rows into a table, use the `executemany()` method.

The second parameter of the `executemany()` method is a list of tuples, containing the data you want to insert:

```
import mysql.connector

mycursor = mydb.cursor()

sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"
val = [
    ('Peter', 'Lowstreet 4'),
    ('Amy', 'Apple st 652'),
    ('Hannah', 'Mountain 21'),
    ('Michael', 'Valley 345'),
    ('Sandy', 'Ocean blvd 2'),
    ('Viola', 'Sideway 1633')
]

mycursor.executemany(sql, val)

mydb.commit()

print(mycursor.rowcount, "was inserted.")
```

Get Inserted ID

You can get the id of the row you just inserted by asking the cursor object. **Note:** If you insert more than one row, the id of the last inserted row is returned.

```
mycursor = mydb.cursor()

sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"
val = ("Michelle", "Blue Village")
mycursor.execute(sql, val)

mydb.commit()

print("1 record inserted, ID:", mycursor.lastrowid)
```

Select From a Table

To select from a table in MySQL, use the "SELECT" statement:

```
mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM customers")
```

```
myresult = mycursor.fetchall()
```

```
for x in myresult:  
    print(x)
```

Note: We use the `fetchall()` method, which fetches all rows from the last executed statement.

```
testCursor.execute(selectQuery)  
queryResults = testCursor.fetchall()
```

```
for eachRow in queryResults:  
    print(eachRow[0])
```

Selecting Columns

To select only some of the columns in a table, use the "SELECT" statement followed by the column name(s):

```
mycursor = mydb.cursor()
```

```
mycursor.execute("SELECT name, address FROM customers")
```

```
myresult = mycursor.fetchall()
```

```
for x in myresult:  
    print(x)
```

Using the fetchone() Method

If you are only interested in one row, you can use the `fetchone()` method. The `fetchone()` method will return the first row of the result:

```
mycursor.execute("SELECT * FROM customers")
```

```
myresult = mycursor.fetchone()
```

```
print(myresult)
```

Select With a Filter

When selecting records from a table, you can filter the selection by using the "WHERE" statement:

```
mycursor = mydb.cursor()

sql = "SELECT * FROM customers WHERE address = 'Park Lane 38'"

mycursor.execute(sql)

myresult = mycursor.fetchall()

for x in myresult:
    print(x)
```

Wildcard Characters

You can also select the records that starts, includes, or ends with a given letter or phrase.

Use the `%` to represent wildcard characters:

```
mycursor = mydb.cursor()

sql = "SELECT * FROM customers WHERE address LIKE '%way%'"

mycursor.execute(sql)

myresult = mycursor.fetchall()

for x in myresult:
    print(x)
```

Prevent SQL Injection

When query values are provided by the user, you should escape the values.

This is to prevent SQL injections, which is a common web hacking technique to destroy or misuse your database.

The `mysql.connector` module has methods to escape query values:

```
sql = "SELECT * FROM customers WHERE address = %s"
adr = ("Yellow Garden 2", )

mycursor.execute(sql, adr)
```

```
myresult = mycursor.fetchall()
```

```
for x in myresult:  
    print(x)
```

Sort the Result

Use the ORDER BY statement to sort the result in ascending or descending order.

The ORDER BY keyword sorts the result ascending by default. To sort the result in descending order, use the DESC keyword.

```
sql = "SELECT * FROM customers ORDER BY name"
```

```
mycursor.execute(sql)
```

```
myresult = mycursor.fetchall()
```

```
for x in myresult:  
    print(x)
```

Delete Record

You can delete records from an existing table by using the "DELETE FROM" statement:

```
sql = "DELETE FROM customers WHERE address = 'Mountain 21'"
```

```
mycursor.execute(sql)
```

```
mydb.commit()
```

```
print(mycursor.rowcount, "record(s) deleted")
```

Important!: Notice the statement: `mydb.commit()`. It is required to make the changes, otherwise no changes are made to the table.

Notice the WHERE clause in the DELETE syntax: The WHERE clause specifies which record(s) that should be deleted. If you omit the WHERE clause, all records will be deleted!

```
sql = "DELETE FROM customers WHERE address = %s"
```

```
adr = ("Yellow Garden 2", )
```

```
mycursor.execute(sql, adr)
```

```
mydb.commit()
```

```
print(mycursor.rowcount, "record(s) deleted")
```

Update Table

You can update existing records in a table by using the "UPDATE" statement:

```
sql = "UPDATE customers SET address = 'Canyon 123' WHERE address = 'Valley 345'"
```

```
mycursor.execute(sql)
```

```
mydb.commit()
```

```
print(mycursor.rowcount, "record(s) affected")
```

It is considered a good practice to escape the values of any query, also in update statements.

```
sql = "UPDATE customers SET address = %s WHERE address = %s"
```

```
val = ("Valley 345", "Canyon 123")
```

```
mycursor.execute(sql, val)
```

```
mydb.commit()
```

```
print(mycursor.rowcount, "record(s) affected")
```

Calling Store Procedure in Python:

```
mycursor.callproc("store_procedure_name")
```

```
for eachResult in mycursor.stored_results():  
    print(eachResult)
```