## **🐍 Python MySQL DB Connectivity Guide**

### **📦 Step 1: Install Required Package**

pip install pymysql

### **⚙️ Step 2: Database Connection Setup**

Use pymysql.connect() to connect to a MySQL database.

import pymysql

connection = pymysql.connect(

host='localhost', # Replace with your DB host

user='root', # Replace with your DB user

password='your\_password', # Replace with your DB password

database='test\_db', # Replace with your DB name

cursorclass=pymysql.cursors.DictCursor # Return rows as dictionaries

)

### **🛠️ Step 3: Create Table (DDL)**

with connection.cursor() as cursor:

create\_query = """

CREATE TABLE IF NOT EXISTS employees (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100),

department VARCHAR(100)

);

"""

cursor.execute(create\_query)

### **📝 Step 4: Insert Data (DML)**

#### **Option 1: Insert One Row**

cursor.execute("INSERT INTO employees (name, department) VALUES (%s, %s)", ("John", "IT"))

#### **Option 2: Insert Multiple Rows (Recommended)**

values = [("John", "IT"), ("Alice", "HR"), ("Bob", "Finance")]

insert\_query = "INSERT INTO employees (name, department) VALUES (%s, %s)"

cursor.executemany(insert\_query, values)

connection.commit() # Important!

### **🔍 Step 5: Select Data (DQL)**

cursor.execute("SELECT \* FROM employees")

results = cursor.fetchall()

### **📤 Step 6: Write Output to a File**

with open("employees\_output.txt", "w") as f:

for row in results:

f.write(f"{row}\n")

print("Data written to employees\_output.txt")

### **✅ Full Code Block**

import pymysql

connection = pymysql.connect(

host='localhost',

user='root',

password='your\_password',

database='test\_db',

cursorclass=pymysql.cursors.DictCursor

)

try:

with connection.cursor() as cursor:

# Create table

cursor.execute("""

CREATE TABLE IF NOT EXISTS employees (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100),

department VARCHAR(100)

);

""")

# Insert data

data = [("John", "IT"), ("Alice", "HR"), ("Bob", "Finance")]

cursor.executemany("INSERT INTO employees (name, department) VALUES (%s, %s)", data)

connection.commit()

# Select data

cursor.execute("SELECT \* FROM employees")

results = cursor.fetchall()

# Write to file

with open("employees\_output.txt", "w") as f:

for row in results:

f.write(f"{row}\n")

print("Data written to employees\_output.txt")

finally:

connection.close()

### **🧠 Bonus: Is This JDBC?**

* ❌ Technically **not JDBC**, which is Java-specific.
* ✅ But this is Python’s **equivalent DB-API approach**, providing:  
  + Driver-based connection
  + Cursor execution
  + SQL query support
  + Transaction handling

**Secure MySQL Password Handling and Connection in Python**

## **📄 Project Description:**

This project demonstrates a secure and production-friendly approach to handling database passwords in Python applications.

It includes:

* 🔐 Password encryption using Fernet (symmetric encryption)
* 🛡️ Safe decryption at runtime with protection against accidental leaks (print/log masking)
* ⚙️ Actual MySQL database connection using the decrypted password
* 📁 Clean modular code organization for real-world project structure

## **🧱 Tech Stack:**

* **Language: Python 3**
* **Database: MySQL**
* **Libraries:**
  + **cryptography – for encryption and decryption**
  + **mysql-connector-python – to connect to MySQL from Python**

## **🗂️ Project Folder Structure:**

secure-mysql-python/

├── password\_utils.py # Handles encryption, decryption, and secure password masking

├── encrypt\_once.py # One-time script to encrypt your MySQL password

├── mysql\_connect\_safe.py # Main script that connects to MySQL securely

├── secret.key # Auto-generated encryption key (never upload this to GitHub)

└── README.md # Project explanation and setup steps

## **🔐 Features:**

| **Feature** | **Description** |
| --- | --- |
| **Encryption** | Password is encrypted using a randomly generated 32-byte Fernet key |
| **Key Management** | The key is generated once and stored in a local file secret.key |
| **Secure Decryption** | Password is decrypted only at runtime using the key |
| **Print-Safe Strings** | Decrypted password uses a custom FakeStr class to prevent print/log leaks |
| **MySQL Integration** | Connects to a MySQL database using the decrypted password |

## **📌 Real-World Use Case:**

This setup mimics what developers and data engineers should follow when building internal tools, automation scripts, or backend jobs that involve sensitive credentials like:

* MySQL / PostgreSQL DBs
* Cloud storage credentials
* Secure API keys

By avoiding plaintext passwords, the project improves security and follows **DevSecOps best practices**.

**Code**

## **✅ Steps to Run the Project:**

1. **Install dependencies**

pip install cryptography mysql-connector-python

### **✅ Step 1: Run a script once to encrypt your MySQL root password**

Create a script called encrypt\_once.py:

from cryptography.fernet import Fernet

def generate\_key():

key = Fernet.generate\_key()

with open("secret.key", "wb") as f:

f.write(key)

print("✅ Key saved to 'secret.key'")

def encrypt\_password(password):

key = open("secret.key", "rb").read()

f = Fernet(key)

encrypted = f.encrypt(password.encode())

print("🔐 Encrypted password to copy:")

print(encrypted)

if \_\_name\_\_ == "\_\_main\_\_":

# Uncomment if running for the first time

# generate\_key()

# Replace with your real MySQL root password

encrypt\_password("your\_mysql\_root\_password\_here")

### **✅ Step 2: Paste the encrypted password inside password\_utils.py**

Here’s your updated password\_utils.py:

from cryptography.fernet import Fernet

class FakeStr(str):

def \_\_str\_\_(self):

return "\*\*\*\*"

def \_\_repr\_\_(self):

return "\*\*\*\*"

def load\_key():

return open("secret.key", "rb").read()

def decrypt\_password(encrypted\_password):

key = load\_key()

f = Fernet(key)

decrypted = f.decrypt(encrypted\_password).decode()

return FakeStr(decrypted)

def get\_decrypted\_password():

encrypted\_password = b'gAAAAABm...==' # 🔐 Paste the encrypted output here

return decrypt\_password(encrypted\_password)

### **✅ Step 3: Use get\_decrypted\_password() in your actual MySQL script**

In mysql\_connect\_safe.py:

import mysql.connector

from password\_utils import get\_decrypted\_password

def connect\_to\_mysql():

conn = mysql.connector.connect(

host="localhost",

user="root",

password=get\_decrypted\_password(),

database="test" # or your own DB

)

print("✅ Connected to MySQL")

conn.close()

if \_\_name\_\_ == "\_\_main\_\_":

connect\_to\_mysql()

### **🧠 Summary**

| **File** | **What goes in there** |
| --- | --- |
| encrypt\_once.py | Raw password (used once) |
| password\_utils.py | Only the encrypted password |
| mysql\_connect\_safe.py | Connects using decrypted password |

This way:

* 👀 No one sees your real root password
* ✅ Password is encrypted
* 🔒 Decrypted only at runtime
* 🛡️ Protected from accidental print with FakeStr

**🎙️ Interview-Style Explanation (Story Format)**

"So, during my free time, I built a mini project to solve a common but often overlooked problem: how to handle database passwords securely in Python applications."

"Typically, in automation scripts or DB tools, developers hardcode the MySQL password like this:"

password = "root" # ❌ bad practice

"This works but is very unsafe — especially when code is shared across teams or checked into version control like Git."

**"So I decided to build a clean, modular solution with these goals:"**

* No hardcoded plain-text passwords in the code
* Password should be stored in encrypted form
* Even if someone prints or logs the password by mistake, it should not be exposed
* Should work just like a normal string when passed into MySQL connection

**"Here's how I solved it:"**

1. **Encryption using Fernet:** I used the cryptography library to generate a secure key (secret.key) and encrypt the MySQL password using symmetric encryption.
2. **One-time Encryption:** I built a script called encrypt\_once.py — where I entered the actual root password just once, and it generated an encrypted version I could safely paste into my code.
3. **Secure Decryption at Runtime:** Then, I created a utility function get\_decrypted\_password() that decrypts the encrypted password using the secret key — but only at runtime.
4. **Accidental Print Protection:** To avoid accidental exposure, I wrapped the decrypted password in a subclass of Python str called FakeStr, which overrides \_\_str\_\_() and \_\_repr\_\_() to return "\*\*\*\*".

So even if someone does:

**print(password)**

**It prints:**

**\*\*\*\***

5. But under the hood, it still behaves like a real password string.

**MySQL Integration:** Finally, I used mysql-connector-python to connect to a MySQL database using the safely decrypted password. I kept everything modular and production-like.

"The result is a very clean and secure way to manage DB credentials that developers can use in real-world projects — especially in automation scripts, backend services, or cron jobs."

## **🔥 Why This Impresses the Interviewer**

* Shows you're security-conscious 🛡️
* Shows you care about code reusability and safety 🔁
* Shows real-world awareness (people DO print passwords accidentally!) 💡
* Shows you know how to integrate Python with databases like MySQL 🧠
* You think beyond “make it work” — you think **"make it safe and clean."**

## **Bonus Ending (if they’re curious)**

"If I had more time, I was planning to extend it to load encrypted credentials from .env files or use a secret manager like AWS Secrets Manager or Vault."

You can even say:

"Would you like me to show you the code or GitHub repo? It’s pretty clean."

### **About the Author**

**Gowtham SB** is a **Data Engineering expert, educator,** **and content creator** with a passion for **big data technologies, as well as cloud and Gen AI** . With years of experience in the field, he has worked extensively with **cloud platforms, distributed systems, and data pipelines**, helping professionals and aspiring engineers master the art of data engineering.

Beyond his technical expertise, Gowtham is a **renowned mentor and speaker**, sharing his insights through engaging content on **YouTube and LinkedIn**. He has built one of the **largest Tamil Data Engineering communities**, guiding thousands of learners to excel in their careers.

Through his deep industry knowledge and hands-on approach, Gowtham continues to **bridge the gap between learning and real-world implementation**, empowering individuals to build **scalable, high-performance data solutions**.

𝐒𝐨𝐜𝐢𝐚𝐥𝐬

🎥𝐘𝐨𝐮𝐓𝐮𝐛𝐞 - https://www.youtube.com/@dataengineeringvideos

📸𝐈𝐧𝐬𝐭𝐚𝐠𝐫𝐚𝐦 - <https://instagram.com/dataengineeringtamil>

📸𝐈𝐧𝐬𝐭𝐚𝐠𝐫𝐚𝐦 - [https://instagram.com/](https://instagram.com/dataengineeringtamil)thedatatech.in

🤝𝐂𝐨𝐧𝐧𝐞𝐜𝐭 𝐟𝐨𝐫 𝟏:𝟏 - https://topmate.io/dataengineering/

💼𝐋𝐢𝐧𝐤𝐞𝐝𝐈𝐧 - https://www.linkedin.com/in/sbgowtham/

🌐𝐖𝐞𝐛𝐬𝐢𝐭𝐞 - https://codewithgowtham.blogspot.com

💻𝐆𝐢𝐭𝐇𝐮𝐛 - http://github.com/Gowthamdataengineer

💬𝐖𝐡𝐚𝐭𝐬 𝐀𝐩𝐩 - https://lnkd.in/g5JrHw8q

📧𝐄𝐦𝐚𝐢𝐥 - atozknowledge.com@gmail.com

📱𝐀𝐥𝐥 𝐌𝐲 𝐒𝐨𝐜𝐢𝐚𝐥𝐬 - <https://lnkd.in/gf8k3aCH>