Python Database Communication (PDBC)
Every application required to store data permanently. Data can be stored permanently using 2 systems.

1. File System
2. Database System

Limitations of File System
1. Files cannot hold large amount data
2. Files cannot provide security because files are managed by operating system
3. Files are application dependent
4. Files cannot provide Query language

To overcome these limitations we use database systems or database application.

Advantage
1. Database can hold large amount data
2. Data stored inside database is secured, which is provided by database system
3. Database is application independent
4. Database provides a query language for manipulating data (SQL)

Database software's

Database software name	**Company**
Oracle	Oracle corporation
MySQL	Oracle corporation
SQLServer	Microsoft
PostgreSQL	PostgreSQL
SQLlite	PSF (Python Software Foundation)
MongoDB	MongoDB

SQL
SQL stands for Structured Query Language. It is a standard developed by ASNI (American National Standard Institute) to communicate with database.

SQL is used by database software for manipulating data.
SQL provides the command to perform operation on database.
SQL commands are classified categories

1. DDL → Data Definition Language
    a. CREATE
    b. ALTER
    c. DROP
2. DML → Data Manipulation Language
    a. INSERT
    b. UPDATE
    c. DELETE
3. DCL → Data Control Language
    a. GRANT
    b. REVOKE
4. TCL → Transaction Control Language
    a. COMMIT
    b. ROLLBACK
5. DRL → Data Reading Language
    a. SELECT

In order to communicate with database, database vendors provides libraries

Database	**Library**
MySQL Database	mysql-connector-python
Oracle Database	oracledb(cx_oracle)
SQL Server	Pymssql
PostgreSQL	psycopg2
MongoDB	Pymongo
SQLite	Sqlite3 (default library which comes with python software)

All these libraries are developed by the rules or abstracts given python called DB-API (database Application Programming Interface). The functions of all these libraries are same but implementation is different.

How to communicate with MySQL Database?

1. Install MySQL database software
2. Install mysql-connector-python library

Install mysql database software

https://dev.mysql.com/downloads/installer/

install mysql-connector-python library

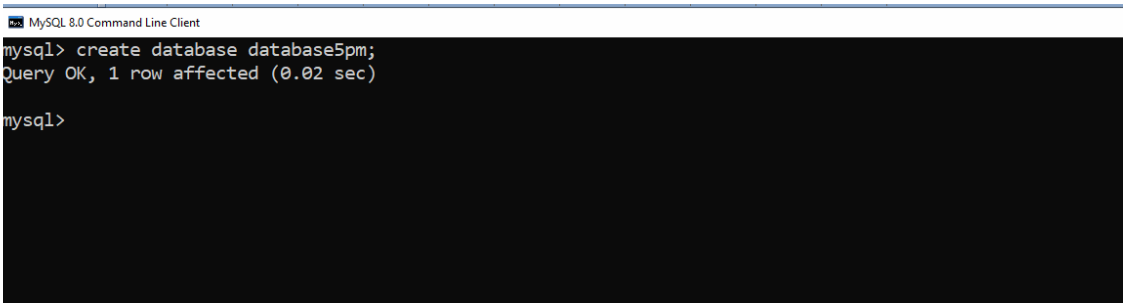
```
C:\Users\Satish Guptha Sir>pip install mysql-connector-python
Requirement already satisfied: mysql-connector-python in c:\users\satish guptha sir\appdata\local\programs\python
\python312\lib\site-packages (9.1.0)

[notice] A new release of pip is available: 24.2 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip

C:\Users\Satish Guptha Sir>
```

Basic steps for communicating with database

1. Establish Connection to Database
2. Create cursor object
3. Sending SQL statements to Database using cursor
4. Reading results from cursor object
5. Closing Connection



connect() function
It is predefined function, this function is used to establish connection to database and return database connection object.

Syntax: connect(database,user,password,host,port)

➔ Database → database5pm
➔ User → root
➔ Password → root

➔ Host → localhost/ip-address
➔ Port → 3306

Example:

```
# Write a program to establish connection to mysql database

import mysql.connector

cn=mysql.connector.connect(database="database5pm",
                           user="root",
                           password="root",
                           host="localhost",
                           port=3306)
print("connection established")
print(cn)
cn.close()
```

Output

```
connection established
<mysql.connector.connection_cext.CMySQLConnection object at
0x000001FBAAEF75F0>
```

Cursor

Cursor object is used for sending SQL statements to database using connection object. After executing SQL statements the result of SQL statements are stored inside cursor object.

Syntax

connection-object-name.cursor()

cursor object provides the following methods to send SQL statements to database

1. execute()
2. executemany()

execute() method send one SQL statement to database
executemany() method send list of SQL statements to database

Example:

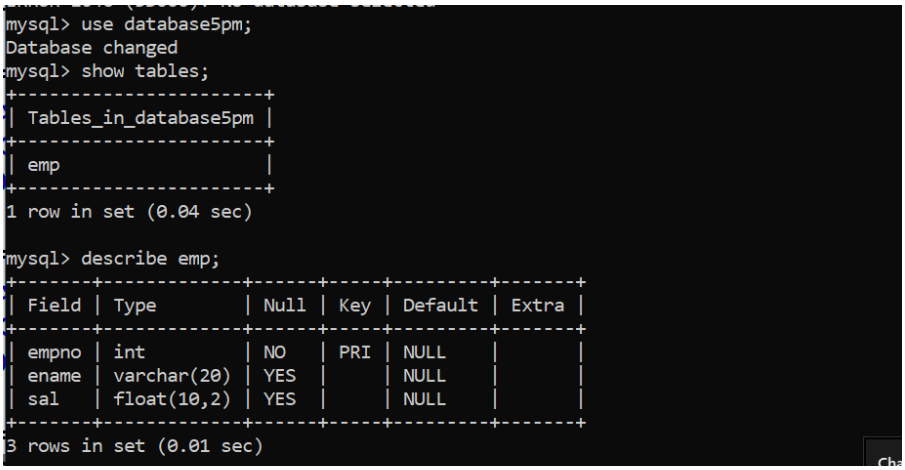
Write a program to create table in database5pm

```
import mysql.connector

cn=mysql.connector.connect(database="database5pm",
                           user="root",
                           password="root",
                           host="localhost",
                           port="3306")
c=cn.cursor()
c.execute('''create table emp(
empno integer(5) primary key,
ename varchar(20),
sal float(10,2))''')
print("Table Created...")
cn.close()
```

Output

Table Created...



Example:

Write a program to insert data into emp table

```
import mysql.connector

cn=mysql.connector.connect(database="database5pm",
                           user="root",
                           password="root",
```

```
                    host="localhost",
                    port="3306")
c=cn.cursor()
while True:
   empno=int(input("EmployeeNo :"))
   ename=input("EmployeeName :")
   sal=float(input("EmployeeSalary :"))
   try:
      c.execute('insert into emp
values(%s,%s,%s)',params=(empno,ename,sal))
      print("employee created")
   except:
      print("error in inserting data")

   ans=input("Add another employee?")
   if ans=="no":
      break


cn.commit()
cn.close()
```

Output

```
EmployeeNo :1
EmployeeName :naresh
EmployeeSalary :4500
employee created
Add another employee?yes
EmployeeNo :2
EmployeeName :suresh
EmployeeSalary :5600
employee created
Add another employee?yes
EmployeeNo :3
EmployeeName :kishore
EmployeeSalary :8900
employee created
Add another employee?yes
EmployeeNo :4
EmployeeName :kiran
```

EmployeeSalary :3500
employee created
Add another employee?yes
EmployeeNo :5
EmployeeName :rajesh
EmployeeSalary :2300
employee created
Add another employee?no

```
mysql> select * from emp;
+-------+---------+---------+
| empno | ename   | sal     |
+-------+---------+---------+
|     1 | naresh  | 4500.00 |
|     2 | suresh  | 5600.00 |
|     3 | kishore | 8900.00 |
|     4 | kiran   | 3500.00 |
|     5 | rajesh  | 2300.00 |
+-------+---------+---------+
5 rows in set (0.00 sec)

mysql>
```

Cursor object provides the following methods

1. fetchone() : returns one row (tuple)
2. fetchmany(n) : returns n rows (list of tuples)
3. fetchall() : returns all rows (list of tuples)

Example:

```
# Write a program to read data from emp table

import mysql.connector
cn=mysql.connector.connect(database="database5pm",
                           user="root",
                           password="root",
                           host="localhost",
                           port="3306")
c=cn.cursor()
c.execute("select * from emp")
row1=c.fetchone()
print(row1)
row2=c.fetchone()
print(row2)
row3=c.fetchone()
print(row3)
row4=c.fetchone()
```

```
print(row4)
row5=c.fetchone()
print(row5)
row6=c.fetchone()
print(row6)
```

Output

```
(1, 'naresh', 4500.0)
(2, 'suresh', 5600.0)
(3, 'kishore', 8900.0)
(4, 'kiran', 3500.0)
(5, 'rajesh', 2300.0)
None
```

```
# Write a program to read data from emp table and calculate
# total salary

import mysql.connector
cn=mysql.connector.connect(database="database5pm",
                           user="root",
                           password="root",
                           host="localhost",
                           port="3306")
c=cn.cursor()
c.execute("select * from emp")
total=0
while True:
    row=c.fetchone()
    if row==None:
        break
    total+=row[2]
    print(f'{row[0]}\t{row[1]}\t{row[2]}')


print(f'Total Salaries {total}')
```

Output

```
1       naresh      4500.0
2       suresh      5600.0
3       kishore     8900.0
4       kiran 3     500.0
```

5 rajesh 2300.0
Total Salaries 24800.0

Example:

```
import mysql.connector
cn=mysql.connector.connect(database="database5pm",
                           user="root",
                           password="root",
                           host="localhost",
                           port="3306")
c=cn.cursor()
c.execute("select * from emp")
rows=c.fetchmany(2)
print(rows)
rows=c.fetchmany(3)
print(rows)
rows=c.fetchmany(5)
print(rows)

cn.close()
```

Output

```
[(1, 'naresh', 4500.0), (2, 'suresh', 5600.0)]
[(3, 'kishore', 8900.0), (4, 'kiran', 3500.0), (5, 'rajesh', 2300.0)]
[]
```

Example:

```
import mysql.connector
cn=mysql.connector.connect(database="database5pm",
                           user="root",
                           password="root",
                           host="localhost",
                           port="3306")
c=cn.cursor()
c.execute("select * from emp")
rows=c.fetchall()
print(rows)

cn.close()
```

Output

[(1, 'naresh', 4500.0), (2, 'suresh', 5600.0), (3, 'kishore', 8900.0), (4, 'kiran', 3500.0), (5, 'rajesh', 2300.0)]

Example:

```
# Write a program to update salary of an employee
import mysql.connector
cn=mysql.connector.connect(database="database5pm",
                           user="root",
                           password="root",
                           host="localhost",
                           port="3306")
c=cn.cursor()
empno=int(input("EmployeeNo :"))
s=float(input("EmployeeSalary :"))
c.execute('"update emp set sal=sal+%s
where empno=%s"',params=(s,empno))
k=c.rowcount
if k==0:
   print("Invalid EmployeeNo")
else:
   print("Salary Updated")

cn.commit()
cn.close()
```

Output

```
EmployeeNo :1
EmployeeSalary :500
Salary Updated

EmployeeNo :5
EmployeeSalary :1000
Salary Updated

EmployeeNo :10
EmployeeSalary :200
Invalid EmployeeNo
```

Example:

Write a program to delete an employee from emp table

```
import mysql.connector
cn=mysql.connector.connect(database="database5pm",
                           user="root",
                           password="root",
                           host="localhost",
                           port="3306")
c=cn.cursor()
empno=int(input("EmployeeNo to Delete :"))
c.execute("delete from emp where empno=%s",params=(empno,))
k=c.rowcount
if k==0:
   print("Invalid EmployeeNo")
else:
   print("Employee Deleted")
cn.commit()
cn.close()
```

Output

```
EmployeeNo to Delete :1
Employee Deleted

EmployeeNo to Delete :5
Employee Deleted

EmployeeNo to Delete :1
Invalid EmployeeNo
```

Communicating with Oracle Database

1. **install oracle software**

https://www.oracle.com/in/downloads/

2. **install oracledb library**

```
C:\Users\Satish Guptha Sir>pip install oracledb
Collecting oracledb
  Downloading oracledb-3.1.1-cp312-cp312-win_amd64.whl.metadata (5.6 kB)
Collecting cryptography>=3.2.1 (from oracledb)
  Downloading cryptography-45.0.4-cp311-abi3-win_amd64.whl.metadata (5.7 kB)
Requirement already satisfied: cffi>=1.14 in c:\users\satish guptha sir\appdata\local\programs\python\python312\l
ib\site-packages (from cryptography>=3.2.1->oracledb) (1.17.1)
Requirement already satisfied: pycparser in c:\users\satish guptha sir\appdata\local\programs\python\python312\li
b\site-packages (from cffi>=1.14->cryptography>=3.2.1->oracledb) (2.22)
Downloading oracledb-3.1.1-cp312-cp312-win_amd64.whl (1.8 MB)
   ---------------------------------------- 1.8/1.8 MB 11.1 MB/s eta 0:00:00
Downloading cryptography-45.0.4-cp311-abi3-win_amd64.whl (3.4 MB)
   ---------------------------------------- 3.4/3.4 MB 10.6 MB/s eta 0:00:00
Installing collected packages: cryptography, oracledb
Successfully installed cryptography-45.0.4 oracledb-3.1.1

[notice] A new release of pip is available: 24.2 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip

C:\Users\Satish Guptha Sir>
```

```
>>> import oracledb
>>>
```