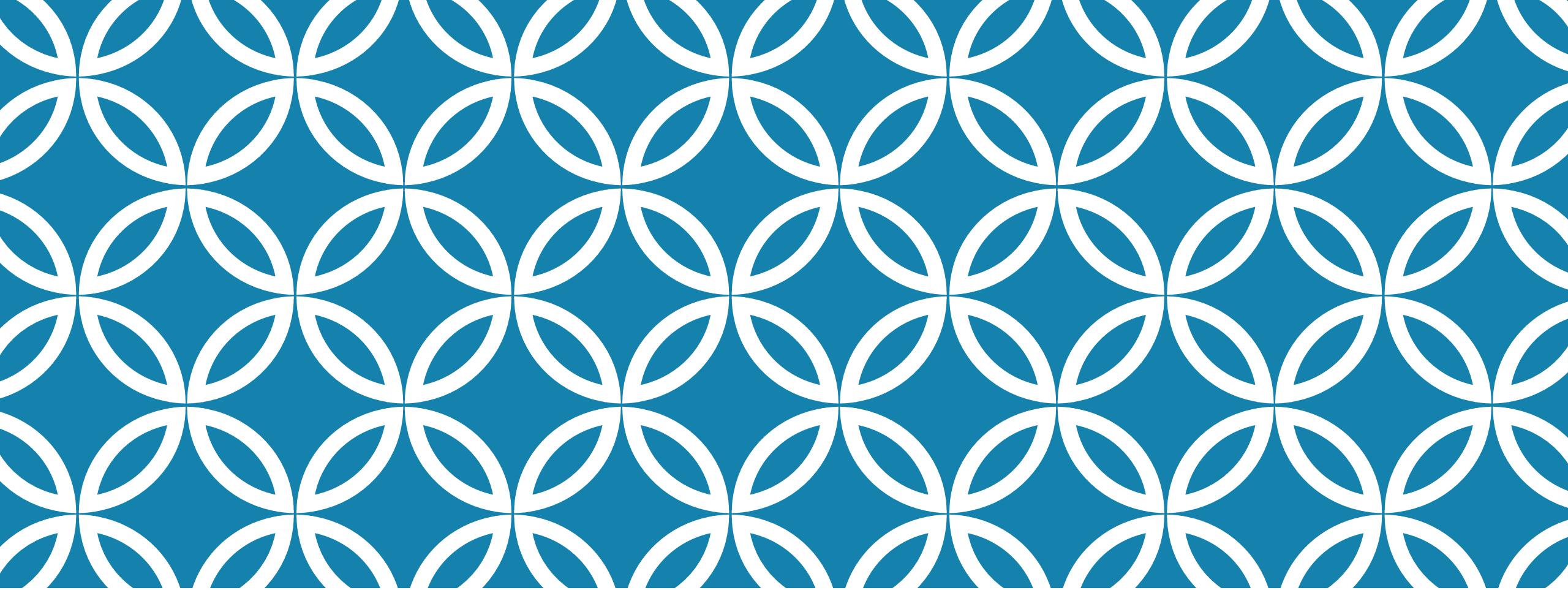
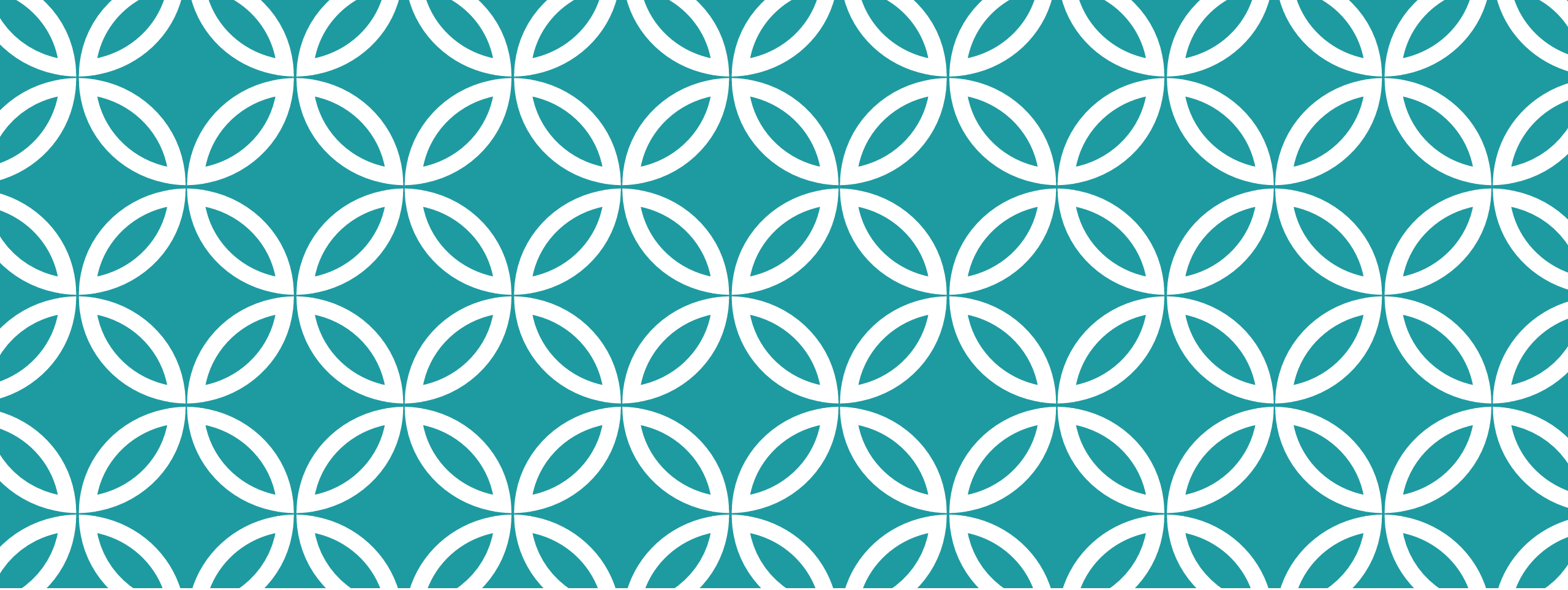


PYTHON



PYTHON WITH MYSQL DATABASE

ENVIRONMENT SETUP

To build the real world applications, connecting with the databases is the necessity for the programming languages. However, python allows us to connect our application to the databases like MySQL, SQLite, MongoDB, and many others.

INSTALL MYSQL.CONNECTOR

To connect the python application with the MySQL database, we must import the mysql.connector module in the program.

The mysql.connector is not a built-in module that comes with the python installation. We need to install it to get it working.

Execute the following command to install it using pip installer.

python -m pip install mysql-connector

DATABASE CONNECTION

There are the following steps to connect a python application to our database.

1. Import mysql.connector module
2. Create the connection object.
3. Create the cursor object
4. Execute the query

CREATING THE CONNECTION

To create a connection between the MySQL database and the python application, the connect() method of mysql.connector module is used.

Pass the database details like HostName, username, and the database password in the method call. The method returns the connection object.

The syntax to use the connect() is given below.

Connection-Object= mysql.connector.connect(
host = <hostname> , user = <username> , passwd = <password>)

EXAMPLE

```
import mysql.connector


#Create the connection object
myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google"
)
#printing the connection object
print(myconn)
```

Here, we must notice that we can specify the database name in the connect() method if we want to connect to a specific database.

```
import mysql.connector


#Create the connection object

myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google"
, database = "mydb")


#printing the connection object

print(myconn)
```

CREATING A CURSOR OBJECT

The cursor object can be defined as an abstraction specified in the Python DB-API 2.0. It facilitates us to have multiple separate working environments through the same connection to the database. We can create the cursor object by calling the 'cursor' function of the connection object. The cursor object is an important aspect of executing queries to the databases.

The syntax to create the cursor object is given below.

<my_cur> = conn.cursor()

```
import mysql.connector

#Create the connection object

myconn = mysql.connector.connect(
host = "localhost", user = "root",passwd = "google", database = "mydb")

#printing the connection object

print(myconn)

#creating the cursor object

cur = myconn.cursor()

print(cur)
```

CREATING NEW DATABASES

Getting the list of existing databases

We can get the list of all the databases by using the following MySQL query.

```
> show databases;
```

```
import mysql.connector

#Create the connection object

myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google"
)

#creating the cursor object

cur = myconn.cursor()

try:

    dbs = cur.execute("show databases")

except:

    myconn.rollback()

for x in cur:

    print(x)

myconn.close()
```

Creating the new database

The new database can be created by using the following SQL query.

> create database <database-name>

CREATING THE TABLE

In this section of the tutorial, we will create the new table Employee. We have to mention the database name while establishing the connection object.

We can create the new table by using the CREATE TABLE statement of SQL. In our database PythonDB, the table Employee will have the four columns, i.e., name, id, salary, and department_id initially.

The following query is used to create the new table Employee.

> create table Employee (name varchar(20) **not** null, id int primary key, salary float **not** null, Dept_Id int **not** null)

ALTER TABLE

Sometimes, we may forget to create some columns, or we may need to update the table schema. The alter statement used to alter the table schema if required. Here, we will add the column branch_name to the table Employee. The following SQL query is used for this purpose.

alter table Employee add branch_name varchar(20) **not** null

INSERT OPERATION

The **INSERT INTO** statement is used to add a record to the table. In python, we can mention the format specifier (%s) in place of values.

We provide the actual values in the form of tuple in the execute() method of the cursor.

Consider the following example.

```
import mysql.connector

myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",
database = "PythonDB")

cur = myconn.cursor()

sql = "insert into Employee(name, id, salary, dept_id, branch_name) values (%s, %s, %s, %s, %s)"

val = ("John", 110, 25000.00, 201, "Newyork")

try:

    cur.execute(sql,val)

    myconn.commit()

except:

    myconn.rollback()

print(cur.rowcount,"record inserted!")

myconn.close()
```

INSERT MULTIPLE ROWS

We can also insert multiple rows at once using the python script. The multiple rows are mentioned as the list of various tuples.

Each element of the list is treated as one particular row, whereas each element of the tuple is treated as one particular column value (attribute).

ROW ID

In SQL, a particular row is represented by an insertion id which is known as row id. We can get the last inserted row id by using the attribute lastrowid of the cursor object.

```
import mysql.connector
#Create the connection object
myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google",database = "PythonDB")

#creating the cursor object
cur = myconn.cursor()

sql = "insert into Employee(name, id, salary, dept_id, branch_name) values (%s, %s, %s, %s, %s)"

val = ("Mike",105,28000,202,"Guyana")
```

```
try:
    #inserting the values into the table
    cur.execute(sql,val)

    #commit the transaction
    myconn.commit()

    #getting rowid
    print(cur.rowcount,"record inserted! id:",cur.lastrowid)

except:
    myconn.rollback()

myconn.close()
```

READ OPERATION

The SELECT statement is used to read the values from the databases. We can restrict the output of a select query by using various clause in SQL like where, limit, etc.

Python provides the fetchall() method returns the data stored inside the table in the form of rows. We can iterate the result to get the individual rows.

In this section of the tutorial, we will extract the data from the database by using the python script. We will also format the output to print it on the console.

```
import mysql.connector

#Create the connection object

myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google"
,database = "PythonDB")

#creating the cursor object

cur = myconn.cursor()

try:

    #Reading the Employee data

    cur.execute("select * from Employee")

 #fetching the rows from the cursor object

    result = cur.fetchall()

    #printing the result
```

```
 for x in result:
        print(x);
except:
    myconn.rollback()
  myconn.close()
```

THE FETCHONE() METHOD

The fetchone() method is used to fetch only one row from the table. The fetchone() method returns the next row of the result-set.

```
import mysql.connector


#Create the connection object
myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "google"
,database = "PythonDB")


#creating the cursor object
cur = myconn.cursor()


try:
    #Reading the Employee data
    cur.execute("select name, id, salary from Employee")
```

```
#fetching the first row from the cursor object
    result = cur.fetchone()

    #printing the result
    print(result)
except:
    myconn.rollback()

myconn.close()
```

UPDATE OPERATION

The UPDATE-SET statement is used to update any column inside the table. The following SQL query is used to update a column.

```
> update Employee set name = 'alex' where id = 110
```

DELETE OPERATION

The DELETE FROM statement is used to delete a specific record from the table. Here, we must impose a condition using WHERE clause otherwise all the records from the table will be removed.

PERFORMING TRANSACTIONS

Transactions ensure the data consistency of the database. We have to make sure that more than one applications must not modify the records while performing the database operations. The transactions have the following properties.

Atomicity
Either the transaction completes, or nothing happens. If a transaction contains 4 queries then all these queries must be executed, or none of them must be executed.

Consistency
The database must be consistent before the transaction starts and the database must also be consistent after the transaction is completed.

Isolation
Intermediate results of a transaction are not visible outside the current transaction.

Durability
Once a transaction was committed, the effects are persistent, even after a system failure.

PYTHON COMMIT() METHOD

Python provides the commit() method which ensures the changes made to the database consistently take place.

The syntax to use the commit() method is given below.

conn.commit()

All the operations that modify the records of the database do not take place until the commit() is called.

PYTHON ROLLBACK() METHOD

The rollback() method is used to revert the changes that are done to the database. This method is useful in the sense that, if some error occurs during the database operations, we can rollback that transaction to maintain the database consistency.

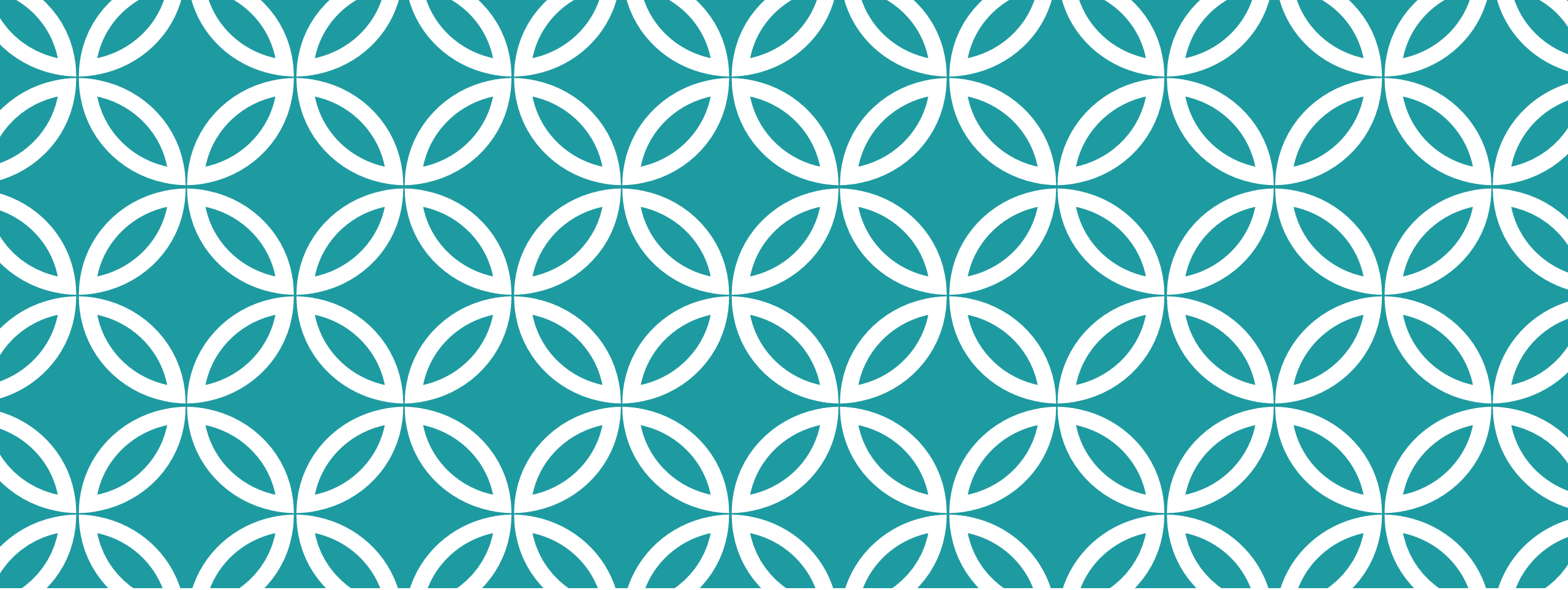
The syntax to use the rollback() is given below.

Conn.rollback()

CLOSING THE CONNECTION

We need to close the database connection once we have done all the operations regarding the database. Python provides the close() method. The syntax to use the close() method is given below.

conn.close()



CONTINUE IN NEXT UNIT .....