Database in PYTHON

INTRODUCTION

In this tutorial you will learn how to use a widely used database management system called MySQL in Python. You do not need any previous knowledge of MySQL to use this tutorial, but there is a lot more to MySQL than covered in this short introductory tutorial.

Installing MySQL

Download and install MySQL from the MySQL's official website. You need to install the MySQL server to follow this tutorial.

Next, you have to install mysql.connector for Python. We need mysql.connector to connect Python Script to the MySQL database.

Now, check whether you have installed the mysql.connector correctly or not using the following code.

```
import mysql.connector
```

If the above code runs without any errors, then you have successfully installed mysql.connector, and it is ready to use.

Connecting and Creating

Now, we will connect to the database using username and password of MySQL. If you don't remember your username or password, create a new user with a password.

```
import mysql.connector as mysql

db = mysql.connect(
    host = "localhost",
    user = "root",
    passwd = “pass"
)

print(db)

# it will print a connection object if everything is fine

<mysql.connector.connection_cext.CMySQLConnection
   object at 0x000020C26A84C50>
```

Creating
DATABASES

To create a database in MySQL, we use CREATE DATABASE database_name statement.

```
# creating an instance of 'cursor' class which is used to execute the 'SQL' statements in 'Python'

cursor = db.cursor()

# creating a databse called ‘pythondb'

# 'execute()' method is used to compile a 'SQL' statement

# below statement is used to create the ‘pythondb' database

cursor.execute("CREATE DATABASE pythondb")
```

SHOW DATABASES

To see all the databases

```
cursor = db.cursor()

# executing the statement using 'execute()' method
   cursor.execute("SHOW DATABASES")

# 'fetchall()' method fetches all the rows from the last
   executed statement

databases = cursor.fetchall() # it returns a list of all
   databases present
```

Now let’s print the list of databases returned below

```
   print(databases)

# showing one by one database

   for database in databases:

            print(database)
```

CREATING Tables

Creating tables in the database to store the information. Before creating tables, we have to select a database first.

Run the following code, to select ***pythondb*** database which we have created a minute before.

```
db = mysql.connect(
    host = "localhost",
    user = "root",
    passwd = "dbms",
    database = “pythondb"
)
```

The above code will execute with no errors if the database exists. Now, you have connected to the database called ***pythondb***.

CREATING Tables

Use the CREATE TABLE table_name to create a table in the selected database.

```
cursor = db.cursor()
# creating a table called 'users' in the ‘pythondb' database
cursor.execute("CREATE TABLE users (name VARCHAR(255), user_name VARCHAR(255))")
```

You have successfully created the table users in the pythondb database. See all the tables present in the database using the SHOW TABLES statement.

SHOW TABLES

To see all the tables in your database

```
cursor = db.cursor()

# executing the statement using 'execute()' method

  cursor.execute("SHOW TABLES")

# 'fetchall()' method fetches all the rows from the last
  executed statement

  tables = cursor.fetchall() # it returns a list of all tables
  present
```

Now let’s print the list of databases returned below

```
  print(tables)

# showing one by one table

  for tab in tables:

          print(tab)
```

Primary KEY

It is a unique value in the table. It helps to find each row uniquely in the table.

To create a Primary Key, we use the PRIMARY KEY statement while creating the table.

The statement INT AUTO_INCREMENT PRIMARY KEY is used to identify each row uniquely with a number starting from 1.

Let's see how to create Primary Key for a table.

PRIMARY KEY

```
cursor = db.cursor()

# first we have to 'drop' the table which was already
  created to create it again with the 'PRIMARY KEY'

# 'DROP TABLE table_name' statement will drop the
  table from a database

cursor.execute("DROP TABLE users")

# creating the 'users' table again with the 'PRIMARY
  KEY'

cursor.execute("CREATE TABLE users (id INT(11) NOT
  NULL AUTO_INCREMENT PRIMARY KEY, name
  VARCHAR(255), user_name VARCHAR(255))")
```

DESC TABLES

To see the structure of the table created

```
cursor = db.cursor()

# 'DESC table_name' is used to get all columns
  information

cursor.execute("DESC users")

# it will print all the columns as 'tuples' in a list

print(cursor.fetchall())
```

Dropping
PRIMARY KEY

We use **ALTER TABLE table_name DROP column_name** statement to drop the column with Primary Key.

```
cursor = db.cursor()


# dropping the 'id' column

cursor.execute("ALTER TABLE users DROP id")

cursor.execute("DESC users")

print(cursor.fetchall())
```

Adding PRIMARY KEY

Adding Primary Key to the existing table. We use **ALTER TABLE table_name ADD PRIMARY KEY(column_name)** statement to add a Primary Key to a table.

```
cursor = db.cursor()

# adding 'id' column to the 'users' table

# 'FIRST' keyword in the statement will add a column in the starting of the table

cursor.execute("ALTER TABLE users ADD COLUMN id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY FIRST")


cursor.execute("DESC users")


print(cursor.fetchall())
```

Inserting Data

Inserting data into table to store it.

Use **INSERT INTO table_name (column_names) VALUES (data)** statement to insert into the table.

Inserting Data

Inserting A Single Row

```
cursor = db.cursor()

# defining the Query

query = "INSERT INTO users (name, user_name)
VALUES (%s, %s)"

# storing values in a variable

values = (“Nnenna", “Amaka")

# executing the query with values

cursor.execute(query, values)

# to make final output we have to run the 'commit()'
method of the database object

db.commit()
```

Inserting Data

Inserting Multiple Rows

To insert multiple rows into the table, we use the executemany() method. It takes a list of tuples containing the data as a second parameter and a query as the first argument.

Inserting Data

Inserting Multiple Rows

```
cursor = db.cursor()
# defining the Query
query = "INSERT INTO users (name, user_name) VALUES (%s, %s)"
# storing values in a variable
values = [
    ("Peter", "peter"),
    ("Amy", "amy"),
    ("Michael", "michael"),
    ("Hennah", "hennah")
]
# executing the query with values
cursor.executemany(query, values)
# to make final output we have to run the 'commit()' method of the database object
db.commit()
print(cursor.rowcount, "records inserted")
```

SELECT Data

To retrieve the data from a table we use, **SELECT column_names FROM table_name** statement.

SELECT DATA

Getting All Records From Table

To get all records from a table, we use * in place of column names. Let's get all the data from the users table which we inserted before.

```
# defining the Query
query = "SELECT * FROM users"

# getting records from the table
cursor.execute(query)

# fetching all records from the 'cursor' object
records = cursor.fetchall()

# Showing the data
for record in records:
    print(record)
```

SELECT DATA

Getting Some Columns

To select some columns from the table mention column name after the SELECT in the statement. Let's retrieve the username column from the users table.

```
# defining the Query

query = "SELECT user_name FROM users"

# getting 'user_name' column from the table

cursor.execute(query)

# fetching all usernames from the 'cursor' object

usernames = cursor.fetchall()

# Showing the data

for username in usernames:

    print(username)
```

SELECT DATA

Where Clause

WHERE is used to select data on some condition. Now, we will select a record with a particular id .

SELECT column_name FROM table_name WHERE condition statement will be used to retrieve the data on some condition.

SELECT DATA

Where Clause

```
# defining the Query
query = "SELECT * FROM users WHERE id = 5"

# getting records from the table
cursor.execute(query)

# fetching all records from the 'cursor' object
records = cursor.fetchall()

# Showing the data
for record in records:
    print(record)
```

SELECT DATA

Order By

Use the ORDER BY to sort the result in ascending or descending order. It sorts the result in ascending order by default, to sort the result in descending order use the keyword DESC.

SELECT column_names FROM table_name ORDER BY column_name statement will be used to sort the result in ascending order by a column.

SELECT column_names FROM table_name ORDER BY column_name DESC statement will be used to sort the result in descending order by a column.

Sorting the data in ascending order using the name column. Let's see the code.

SELECT DATA

Order By

```
# defining the Query
query = "SELECT * FROM users ORDER BY name"

# getting records from the table
cursor.execute(query)

# fetching all records from the 'cursor' object
records = cursor.fetchall()

# Showing the data
for record in records:
    print(record)
```

Delete

DELETE keyword is used to delete the records from the table.

DELETE FROM table_name WHERE condition statement is used to delete records. If you don't specify the condition, then all of the records will be deleted.

Let's delete a record from the users table with any id.

Delete

```
# defining the Query
query = "DELETE FROM users WHERE id = 5"
# executing the query
cursor.execute(query)
# final step to tell the database that we have changed the table data
db.commit()
```

Update

UPDATE keyword is used to update the data of a record or records.

UPDATE table_name SET column_name = new_value WHERE condition statement is used to update the value of a specific row.

Let's update the name of the 1st record.

Update

```
# defining the Query

query = "UPDATE users SET name = ‘field' WHERE id = 1"

# executing the query

cursor.execute(query)

# final step to tell the database that we have changed the table data

db.commit()
```