# Лабораторная работа № 11

**PostgreSQL – интерфейс Python**

PostgreSQL может быть интегрирован с Python с помощью модуля psycopg2. sycopg2 – это адаптер базы данных PostgreSQL для языка программирования Python. psycopg2 был написан с целью быть очень маленьким, быстрым и стабильным как скала. Вам не нужно устанавливать этот модуль отдельно, потому что он поставляется по умолчанию вместе с Python версии 2.5.x и выше.

Чтобы использовать модуль psycopg2, вы должны сначала создать объект Connection, который представляет базу данных, а затем при желании вы можете создать объект курсора, который поможет вам в выполнении всех операторов SQL.

**API модуля Python psycopg2**

Ниже приведены важные подпрограммы модуля psycopg2, которые могут удовлетворить ваши требования по работе с базой данных PostgreSQL из вашей программы Python. Если вы ищете более сложное приложение, вы можете посмотреть официальную документацию модуля Python psycopg2.

|  |  |
| --- | --- |
| **С. Нет.** | **API и описание** |
| 1 | **psycopg2.connect (database = “testdb”, user = “postgres”, password = “cohondob”, host = “127.0.0.1”, port = “5432”)**  Этот API-интерфейс открывает соединение с базой данных PostgreSQL. Если база данных открыта успешно, она возвращает объект подключения. |
| 2 | **connection.cursor ()**  Эта подпрограмма создает **курсор,** который будет использоваться при программировании вашей базы данных на Python. |
| 3 | **cursor.execute (sql [, необязательные параметры])**  Эта подпрограмма выполняет инструкцию SQL. Оператор SQL может быть параметризован (т. Е. Заполнители вместо литералов SQL). Модуль psycopg2 поддерживает заполнитель с использованием знака% s  Например: cursor.execute («вставить в людей значения (% s,% s)», (кто, возраст)) |
| 4 | **cursor.executemany (sql, seq\_of\_parameters)**  Эта подпрограмма выполняет команду SQL для всех последовательностей параметров или отображений, найденных в последовательности sql. |
| 5 | **cursor.callproc (procname [, параметры])**  Эта процедура выполняет хранимую процедуру базы данных с заданным именем. Последовательность параметров должна содержать одну запись для каждого аргумента, ожидаемого процедурой. |
| 6 | **cursor.rowcount**  Этот атрибут только для чтения, который возвращает общее количество строк базы данных, которые были изменены, вставлены или удалены последним последним выполнением \* (). |
| 7 | **connection.commit ()**  Этот метод фиксирует текущую транзакцию. Если вы не вызываете этот метод, все, что вы сделали после последнего вызова commit (), не будет видно из других соединений с базой данных. |
| 8 | **connection.rollback ()**  Этот метод откатывает любые изменения в базе данных с момента последнего вызова commit (). |
| 9 | **connection.close ()**  Этот метод закрывает соединение с базой данных. Обратите внимание, что это не вызывает автоматически commit (). Если вы просто закроете соединение с базой данных без предварительного вызова commit (), ваши изменения будут потеряны! |
| 10 | **cursor.fetchone ()**  Этот метод извлекает следующую строку из набора результатов запроса, возвращая одну последовательность или None, если больше нет данных. |
| 11 | **cursor.fetchmany ([размер = cursor.arraysize])**  Эта процедура извлекает следующий набор строк результата запроса, возвращая список. Пустой список возвращается, когда больше нет доступных строк. Метод пытается извлечь столько строк, сколько указано параметром size. |
| 12 | **cursor.fetchall ()**  Эта процедура извлекает все (оставшиеся) строки результата запроса, возвращая список. Пустой список возвращается, когда нет доступных строк. |

**psycopg2.connect (database = “testdb”, user = “postgres”, password = “cohondob”, host = “127.0.0.1”, port = “5432”)**

Этот API-интерфейс открывает соединение с базой данных PostgreSQL. Если база данных открыта успешно, она возвращает объект подключения.

**connection.cursor ()**

Эта подпрограмма создает **курсор,** который будет использоваться при программировании вашей базы данных на Python.

**cursor.execute (sql [, необязательные параметры])**

Эта подпрограмма выполняет инструкцию SQL. Оператор SQL может быть параметризован (т. Е. Заполнители вместо литералов SQL). Модуль psycopg2 поддерживает заполнитель с использованием знака% s

Например: cursor.execute («вставить в людей значения (% s,% s)», (кто, возраст))

**cursor.executemany (sql, seq\_of\_parameters)**

Эта подпрограмма выполняет команду SQL для всех последовательностей параметров или отображений, найденных в последовательности sql.

**cursor.callproc (procname [, параметры])**

Эта процедура выполняет хранимую процедуру базы данных с заданным именем. Последовательность параметров должна содержать одну запись для каждого аргумента, ожидаемого процедурой.

**cursor.rowcount**

Этот атрибут только для чтения, который возвращает общее количество строк базы данных, которые были изменены, вставлены или удалены последним последним выполнением \* ().

**connection.commit ()**

Этот метод фиксирует текущую транзакцию. Если вы не вызываете этот метод, все, что вы сделали после последнего вызова commit (), не будет видно из других соединений с базой данных.

**connection.rollback ()**

Этот метод откатывает любые изменения в базе данных с момента последнего вызова commit ().

**connection.close ()**

Этот метод закрывает соединение с базой данных. Обратите внимание, что это не вызывает автоматически commit (). Если вы просто закроете соединение с базой данных без предварительного вызова commit (), ваши изменения будут потеряны!

**cursor.fetchone ()**

Этот метод извлекает следующую строку из набора результатов запроса, возвращая одну последовательность или None, если больше нет данных.

**cursor.fetchmany ([размер = cursor.arraysize])**

Эта процедура извлекает следующий набор строк результата запроса, возвращая список. Пустой список возвращается, когда больше нет доступных строк. Метод пытается извлечь столько строк, сколько указано параметром size.

**cursor.fetchall ()**

Эта процедура извлекает все (оставшиеся) строки результата запроса, возвращая список. Пустой список возвращается, когда нет доступных строк.

**ЗАДАНИЕ**

**Подключение к базе данных**

1. Создать (pgadmin4) тестовую базу с именем testdb (create database …)
2. В PC PyCharmCommunity создать новый проект z:\pythonProject
3. Подключить Python Packages psycopg2 к проекту.

Следующий код Python показывает, как подключиться к существующей базе данных. Если база данных не существует, она будет создана и, наконец, будет возвращен объект базы данных.

import psycopg2

conn = psycopg2.connect(database="testdb", user = "postgres", password = "postpass", host = "127.0.0.1", port = "5432")

print ("Opened database successfully")

Здесь вы также можете указать базу данных **testdb в** качестве имени, и если база данных будет успешно открыта, она выдаст следующее сообщение:

Open database successfully

**Создать таблицу**

Следующая программа Python будет использоваться для создания таблицы в ранее созданной базе данных –

import psycopg2

conn = psycopg2.connect(database = "testdb", user = "postgres", password = "postpass", host = "127.0.0.1", port = "5432")

print ("Opened database successfully")

cur = conn.cursor()

cur.execute('''CREATE TABLE COMPANY

(ID INT PRIMARY KEY NOT NULL,

NAME TEXT NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR(50),

SALARY REAL);''')

print ("Table created successfully")

conn.commit()

conn.close()

Когда вышеуказанная программа будет выполнена, она создаст таблицу COMPANY в вашем **test.db** и отобразит следующие сообщения:

Opened database successfully

Table created successfully

**ВСТАВИТЬ Операция**

Следующая программа на Python показывает, как мы можем создавать записи в нашей таблице COMPANY, созданной в приведенном выше примере.

import psycopg2

conn = psycopg2.connect(database = "testdb", user = "postgres", password = "postpass", host = "127.0.0.1", port = "5432")

print "Opened database successfully"

cur = conn.cursor()

cur.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \

VALUES (1, 'Paul', 32, 'California', 20000.00 )");

cur.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \

VALUES (2, 'Allen', 25, 'Texas', 15000.00 )");

cur.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \

VALUES (3, 'Teddy', 23, 'Norway', 20000.00 )");

cur.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \

VALUES (4, 'Mark', 25, 'Rich-Mond ', 65000.00 )");

cur.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \

VALUES (5, 'Alex', 38, 'California', 25000.00 )");

conn.commit()

print ("Records created successfully");

conn.close()

Когда указанная выше программа будет выполнена, она создаст указанные записи в таблице COMPANY и отобразит следующие две строки:

Opened database successfully

Records created successfully

**4). Зайдите в Postgres (pgadmin4) и просмотрите созданную таблицу.**

**ВЫБЕРИТЕ Операцию**

Следующая программа на Python показывает, как мы можем получать и отображать записи из нашей таблицы COMPANY, созданной в приведенном выше примере.

import psycopg2

conn = psycopg2.connect(database = "testdb", user = "postgres", password = "postpass", host = "127.0.0.1", port = "5432")

print ("Opened database successfully")

cur = conn.cursor()

cur.execute("SELECT id, name, address, salary from COMPANY")

rows = cur.fetchall()

for row in rows:

print ("ID = ", row[0])

print ("NAME = ", row[1])

print ("ADDRESS = ", row[2])

print ("SALARY = ", row[3], "\n")

print ("Operation done successfully");

conn.close()

Когда вышеуказанная программа будет выполнена, она даст следующий результат –

Opened database successfully

ID = 1

NAME = Paul

ADDRESS = California

SALARY = 20000.0

ID = 2

NAME = Allen

ADDRESS = Texas

SALARY = 15000.0

ID = 3

NAME = Teddy

ADDRESS = Norway

SALARY = 20000.0

ID = 4

NAME = Mark

ADDRESS = Rich-Mond

SALARY = 65000.0

Operation done successfully

**ОБНОВЛЕНИЕ Операция**

Следующий код Python показывает, как мы можем использовать инструкцию UPDATE, чтобы обновить любую запись, а затем извлечь и отобразить обновленные записи из нашей таблицы COMPANY –

import psycopg2

conn = psycopg2.connect(database = "testdb", user = "postgres", password = "postpass", host = "127.0.0.1", port = "5432")

print "Opened database successfully"

cur = conn.cursor()

cur.execute("UPDATE COMPANY set SALARY = 25000.00 where ID = 1")

conn.commit()

print "Total number of rows updated :", cur.rowcount

cur.execute("SELECT id, name, address, salary from COMPANY")

rows = cur.fetchall()

for row in rows:

print ("ID = ", row[0])

print ("NAME = ", row[1])

print ("ADDRESS = ", row[2])

print ("SALARY = ", row[3], "\n")

print ("Operation done successfully");

conn.close()

Когда вышеуказанная программа будет выполнена, она даст следующий результат –

Opened database successfully

Total number of rows updated : 1

ID = 1

NAME = Paul

ADDRESS = California

SALARY = 25000.0

ID = 2

NAME = Allen

ADDRESS = Texas

SALARY = 15000.0

ID = 3

NAME = Teddy

ADDRESS = Norway

SALARY = 20000.0

ID = 4

NAME = Mark

ADDRESS = Rich-Mond

SALARY = 65000.0

Operation done successfully

**УДАЛЕНИЕ Операция**

Следующий код Python показывает, как мы можем использовать инструкцию DELETE, чтобы удалить любую запись, а затем извлечь и отобразить остальные записи из нашей таблицы COMPANY –

SAPE

SAPE

import psycopg2

conn = psycopg2.connect(database = "testdb", user = "postgres", password = "postpass", host = "127.0.0.1", port = "5432")

print "Opened database successfully"

cur = conn.cursor()

cur.execute("DELETE from COMPANY where ID=2;")

conn.commit()

print "Total number of rows deleted :", cur.rowcount

cur.execute("SELECT id, name, address, salary from COMPANY")

rows = cur.fetchall()

for row in rows:

print "ID = ", row[0]

print "NAME = ", row[1]

print "ADDRESS = ", row[2]

print "SALARY = ", row[3], "\n"

print "Operation done successfully";

conn.close()

|  |  |  |
| --- | --- | --- |
| Отобразите c 5-го по 10-го сотрудников из таблицы «Сотрудники».  declare  cursor empcur is  select employee\_id, first\_name  from employees;    begin  for emprec in empcur  loop  if empcur%rowcount > 4 then  dbms\_output.put\_line( emprec.first\_name);  exit when empcur%rowcount > 10;  end if;  end loop;    end;  Показать должность и имя сотрудника, который работает первый день.  declare  cursor jobscur is select job\_id, job\_title from jobs;  v\_name employees.first\_name%type;  begin  for jobrec in jobscur  loop  select first\_name into v\_name  from employees  where hire\_date = ( select min(hire\_date) from employees where job\_id = jobrec.job\_id)  and job\_id = jobrec.job\_id;    dbms\_output.put\_line( jobrec.job\_title || '-' || v\_name);  end loop;  end;  Обновление зарплаты сотрудника в зависимости от отдела и процента комиссии. Если отдел - 40, увеличьте зарплату на 10%. Если отдел - 70, то на 15%, если комиссия больше 0,3%, то 5%, в противном случае 10%.  declare  cursor empcur is  select employee\_id, department\_id, commission\_pct  from employees;    v\_hike number(2);  begin  for emprec in empcur  loop  if emprec.department\_id = 40 then  v\_hike := 10;  elsif emprec.department\_id = 70 then  v\_hike := 15;  elsif emprec.commission\_pct > 0.30 then  v\_hike := 5;  else  v\_hike := 10;  end if;  update employees set salary = salary + salary \* v\_hike/100  where employee\_id = emprec.employee\_id;    end loop;  end;  Создайте функцию, которая принимает идентификатор менеджера и возвращает имена сотрудников, которые отчитываются перед этим менеджером. Имена должны быть возвращены в виде строки с запятыми, разделяющими имена.  create or replace function get\_employees\_for\_manager(manager number)  return varchar2  is  v\_employees varchar2(1000) := '';  cursor empcur is  select first\_name from employees  where manager\_id = manager;  begin  for emprec in empcur  loop  v\_employees := v\_employees || ',' || emprec.first\_name;  end loop;  -- remove extra , at the beginning  return ltrim(v\_employees,',');  end;  ПРОВЕРЬТЕ РЕЗУЛЬТАТ РАБОТЫ ФУНКЦИИ | PostgreSQL  CREATE OR REPLACE FUNCTION k\_funk()  RETURNS SETOF employees AS  $func$  DECLARE  \_cursor CURSOR FOR SELECT \* FROM employees;  begin  OPEN \_cursor;  MOVE FORWARD 4 IN \_cursor;  RETURN QUERY FETCH 6 FROM \_cursor;  END  $func$ LANGUAGE plpgsql;  BEGIN;  SELECT k\_funk()  COMMIT; | Phyton |