Lab Session 4 - Building Database-Driven Applications

By Edmund Sadgrove

University of New England

Reading

- Chapter 10 from ***Fundamentals of Database Systems*** by Elmazri and Navathe

---

Summary

- Introduction
- House Keeping
- Python3 Revision
- Using psycopg2
- Some Exercises for You
- Using psycopg2 to Create Records

Introduction

- Todays practical session is all about connecting a python script up to our database (hosted using the PostgreSQL DBMS)to build database-driven applications.
- A *Database* driven application is basically any program that uses a centralised database to manage the storage of any form of persistent data.
- This is achieved by connecting your python application up to the database on the DBMS and executing SQL statements to retrieve and manipulate the data within.

House Keeping

- In order to connect your application up to the database, you will need to specify the username and password of your apps account.
- Up until this point in the course, you have been connecting to the DBMS using the `psql` client, which automatically connects you using your UNE username and password.
- To avoid having to specify your UNE username and password in plain-text within your python programs, we have created a new user account in the PosgreSQL sever on turing for each student dedicated for use in your applications.
    - The username for your apps account with by in form: `<your_une_username>_apps`
    - The password for the account will be your **UNE Student Number**

- For example, if my UNE username was *student1* and my student number was *122334455*, my apps account username will by **'student1_apps'** and the password for the account will be **'122334455'**
- You can create databases and log into your apps account using the `createdb` application and 'psql' client, however you will need to explicitly specify the username to stop the client from automatically logging in using you UNE account credentials.
- This is done using the `-U`, `-W` `-h` options:
  - `-U` Specifies the account username that you will be logging in with
  - `-W` Forces the `psql` client to prompt for the password
  - `-h` Sets the host - `127.0.0.1` is turing.une.edu.au
- For todays practical session, you will need to create a new database using you apps account called `<your_username>_apps_prac_5`
- Like this (Sub in your username and enter your student number as the password when prompted)

```
[mwelch8@turing ~]$ createdb -U mwelch8_apps -W -h 127.0.0.1 mwelch8_apps_prac_5
Password:
[mwelch8@turing ~]$ psql -U mwelch8_apps -W -h 127.0.0.1 mwelch8_apps_prac_5
Password for user mwelch8_apps:
psql (9.4.4)
Type "help" for help.

mwelch8_apps_prac_5=⇒ \dt
No relations found.
mwelch8_apps_prac_5=⇒
```

- Once you have successfully logged into to your newly created prac_05 database using your apps account, you will need to build your COMPANY database.
- We can do this by importing a pg_dump of practical ones database, we will call this prac_01.sql.
- Before we can do this, we need to change the owners name to _apps, as this is your apps account which is a different account to your standard psql account. This is required to access our database from our application.
- The easiest way to do this is to use a text editor such as gedit and use the find and replace command to replace all instances of your username. If you miss this step you can use dropdb with the syntax below to drop the db and start again.

```
[mwelch8@turing ~]$ createdb -U mwelch8_apps -W -h 127.0.0.1 mwelch8_apps_prac_5
[mwelch8@turing ~]$ psql -U mwelch8_apps -W -h 127.0.0.1 mwelch8_apps_prac_5
psql (9.4.4)
Type "help" for help.

mwelch8_apps_prac_5=⇒ \i ~/prac_01.sql

 ...

mwelch8_apps_prac_5=⇒ \dt
             List of relations
 Schema |      Name       | Type  |  Owner
--------+-----------------+-------+----------
 public | department      | table | mwelch8
 public | dependent       | table | mwelch8
 public | dept_locations  | table | mwelch8
 public | employee        | table | mwelch8
 public | project         | table | mwelch8
 public | works_on        | table | mwelch8
(6 rows)
```

- **NOTE** Running the sql script in this way may produce some errors when you run this script. These errors are from the commands that attempt to change the owner of the relations to account under which they were

originally created.

- These can be ignored as we want the owner of these relations to default to the current account.
- Now you have your database for todays practical session set up.

Python3 Revision

- As quick revision exercise we will review, compile and run a couple of simple Python programs just to get everyone up to speed.
- The first program will look at is the ubiquitous `Hello World` exmaple:

```
"""************************************************************************
 *  Execution:    python3 HelloWorld.py
 *
 *  Prints "Hello, World". By tradition, this is everyone's first program.
 *
 *  % python3 HelloWorld.py
 *  Hello, World
 *
 ************************************************************************"""

def main():
    print("Hello, World")

if __name__ == "__main__":
    main()
```

- Recall that we can run a Python script by using the python command at the bash shell.

```
21641:prac_5 mwelch8$ python3 HelloWorld.py
Hello, World
21641:prac_5 mwelch8$
```

- For processing user input from the keyboard we can use the `Input` function.

```
"""*********************************************************
* A python script that reads linea from standard input
* input:
*  input() - reads a line of input from the stream
*********************************************************"""

def main():
    print("Welcome to my first python script")

    try:
        firstName = input("Please Enter In Your First Name: ")
        bornYear = input("Please Enter In The Year You Were Born: ")
        thisYear = input("Please Enter In The Current Year: ")

        bYear = int(bornYear)
        tYear = int(thisYear)

        age = tYear-bYear
        print("Hello ",firstName, " You are ", age, " years old")
    except:
        print("Error reading line")

if __name__ == "__main__":
    main()
```

- This code was adapted from a tutorial available at:http://www.codeproject.com/Articles/2853/Java-Basics-Input-and-Output

- Compile and run these examples and the code.

Using psycopg2

- Now we are ready to write a Python program that connects to our database.
- For this we can use psycopq2, which should be installed on turing.
- Now you are ready to run your first database connected through a Python program.
- The first program that we will look at connects to your COMPANY database and lists the first and last names of all employees:

```
import psycopg2

def main():
    conn = None
    try:
       #connect to database
       conn = psycopg2.connect(
              dbname='<user_name>_apps_prac_5',
              user='<user_name>_apps',
              host='127.0.0.1',
              password='<student_number>')

       #create database cursor
       cur = conn.cursor()

       #execute statement allowing execute method to sanitize
       cur.execute("SELECT fname,lname from employee")

       #fetch results
       results = cur.fetchall()

       #print rows row by row
       print("\n****Employees Currently Within the Database*****:\n")
       for row in results:
           print("   ", row)
       print("Query Executed Successfully ... exiting")

    except (Exception, psycopg2.DatabaseError) as error:
       print("Error connecting to database: ", error)

    finally:
        if(conn):
            cur.close()
            conn.close()
            print("Connection closed")

if __name__ == "__main__":
    main()
```

- To test this program, copy and paste the code above and save it to a file called DbTester.py
- Before you can run this program, you will need to enter the details in for the database for the database connection in the lines shown below:

```
conn = psycopg2.connect(
              dbname='<user_name>_apps_prac_5',
              user='<user_name>_apps',
              host='127.0.0.1',
              password='<student_number>')
```

- The dbname will need to list the database that you are attempting to connect to - in this situation it will be the prac_5 database that you created earlier in the prac.

- In the call to the `psycopg2.connect( … )`, you will also need to specify the name of your apps account (which will be in the form "\<une_username>_apps") and the password for this account (which we have set to be your UNE student number).
- Once you have entered these items, you can run the program with python3. If you have correctly set your database up and updated the details in the source-code, the output should look something like this:

```
[mwelch8@turing prac_5]$ python3 DbTester.py

****Employees Currently Within the Database*****

First Name   Last Name
--------------------
Alex         Freed
Bob          Bender
Evan         Wallis
James        Borg
Jared        James
John         James
Kim          Grace
Ahmad        Jabbar
Alicia       Zelaya
Franklin     Wong
Jennifer     Wallace
Red          Bacher
Sammy        Hall
Carl         Reedy
Naveen       Drew
Ray          King
Billie       King
Jon          Kramer
Arnold       Head
Gerald       Small
Helga        Pataki
Lyle         Leslie
Jill         Jarvis
Kate         King
Nandita      Ball
Alec         Best
Bonnie       Bays
Sam          Snedden
John         Smith
Joyce        English
Ramesh       Narayan
Jeff         Chase
Chris        Carter
Jenny        Vos
Andy         Vile
Josh         Zell
Tom          Brand
Brad         Knight
Jon          Jones
Justin       Mark
--------------------

Query Executed Successfully … exiting
[mwelch8@turing prac_5]$
```

- Now lets walk though our example:
- The first section sets the database cursor:

```
cur = conn.cursor()
```

- The database cursor will point to the current row in the database from a query
- We then specify the SQL query we want to execute on the database as a string and execute the query. In this example, we are simply selecting the `fname` and `lname` columns for all employees.

- It is important we let the execute method sanitize our results, for this reason any external string formatting will be avoided.

```
cur.execute("SELECT fname,lname from employee")
```

- We then fetch the results of our query.

```
results = cur.fetchall()
```

- The results returned from the DBMS are stored in the `result` list, with ech row stored as an item that we can iterate through using a for loop:

```
for row in results:
           print("   ", row)
```

Exercises for You

1. Modify the `DbTester` program so that it displayes the `fname`, `lname`, `bdate`, `sex` and salary from the employee table in a formatted list with columns that are spaced with appropriate column headings.
    - To achieve this you will need to modify the query your program is executing and return the columns required.
    - You will need to add a nested for loop to read through each column attribute e.g. row[j].
2. Modify the `DbTester` program so that it displays the department name (`dname`)and department location (`dlocation`) in addition to the attributes displayed in question 1.

Using psycopg2 to Create Records

- Now that we have Python program connecting up and bringing data down form our database, we can look at developing a Python program that modifies some of your data.
- First we will look at a simple example that creates a new record in the employee table.

```
import psycopg2

def main():
    #The query template, the list returns parentheses ( ... ), so we don't need them for VALUES( ... )
    sql = """INSERT INTO employee(fname, minit,lname,ssn,dno) VALUES%s;"""
    #create a row list to insert
    insert_employee_list = ("new_fname","S","new_lname", 112233446, 5)
    conn = None
    try:
       #connect to database
       conn = psycopg2.connect(dbname='<user_name>_apps_prac_5',
                               user='<user_name>_apps',
                               host='127.0.0.1',
                               password='<student_number>')
       #create database cursor
       cur = conn.cursor()

       #execute statement allowing execute to sanitize
       cur.execute(sql,(insert_employee_list,))

       #commit changes
       conn.commit()

       print("Query Executed Successfully ... exiting")

    except (Exception, psycopg2.DatabaseError) as error:
       print("Error connecting to database: ", error)

    finally:
```

```
        if(conn):
            cur.close()
            conn.close()
            print("Connection closed")

if __name__ == "__main__":
    main()
```

- After running this program on your apps database:

```
mwelch8_apps_prac_5⇒ select fname,lname from employee;

...

 Jon       | Jones
 Justin    | Mark
 New_fname | New_lname
(41 rows)

mwelch8_apps_prac_5⇒
```

- This program connects to our database using the same process as in the first example.
- The new code sits within the central `try:` block.
- We have added an sql template and a list to store our new row:

```
sql = """INSERT INTO employee(fname, minit,lname,ssn,dno) VALUES%s;"""
insert_employee_list = ("new_fname","S","new_lname", 112233446, 5)

try:
   cur.execute(sql,(insert_employee_list,))

   conn.commit()
```

- Here we create a new row from the list that we initialize first.
- We then use the `execute` method to format and sanitize our string query from the template.
- Finally, we call the `commit()` method to confirm the changes to the database.
- What happens when you run this program a second time? (Connected to the same database)
    - Hint: Can you insert records into a database with duplicate primary keys?

Exercises for You

1. Update the following program to insert the employee information entered through the console session.

```
import psycopg2

def main():

    """- Employee table. Here to show column list
    CREATE TABLE employee (
    fname character varying(15) NOT NULL,
    minit character varying(1),
    lname character varying(15) NOT NULL,
    ssn character(9) NOT NULL,
    bdate date,
    address character varying(50),
    sex character(1),
    salary numeric(10,2),
```

```
      superssn character(9),
      dno integer
      );
      """

      try:
          firstName = input("Please Enter the employee's First Name: ")
          minit = input("Please Enter the employee's middle initial: ")
          lastName = input("Please Enter the employee's Last Name: ")
          ssn = input("Please Enter the employee's Ssn: ")
          dno = input("lease Enter the employee's Department Number: ")

          dno_int = int(dno)

          # Add the additional data feilds here

      except (IOError, ValueError):
          print("Error reading input")

      print("\n****Inserting a New Employee*****\n");

      # The query template.
      # The list returns parentheses ( ... ), so we don't need them for VALUES( ... )
      sql = """INSERT INTO employee(fname, minit,lname,ssn,dno) VALUES%s;"""

      # Create a row list to insert
      # This is where you will need to update the code to include
      # the data entered by the user.

      insert_employee_list = ( ... , .. ,  ... ,  ... ,  ... )

      conn = None
      try:
         # Connect to database
         conn = psycopg2.connect(dbname='<user_name>_apps_prac_5',
                                 user='<user_name>_apps',
                                 host='127.0.0.1',
                                 password='<student_number>')
         # Create database cursor
         cur = conn.cursor()

         # Execute statement allowing execute to sanitize
         cur.execute(sql,(insert_employee_list,))

         # Commit changes
         conn.commit()

         print("Query Executed Successfully ... exiting")

      except (Exception, psycopg2.DatabaseError) as error:
         print("Error connecting to database: ", error)
      # If it is successful or it fails, always close connection
      finally:
          if(conn):
              cur.close()
              conn.close()
              print("Connection closed")



  if __name__ == "__main__":
      main()
```