**РОБОТА З PYTHON POSTGRESQL В СЕРЕДОВИЩІ WINDOWS**

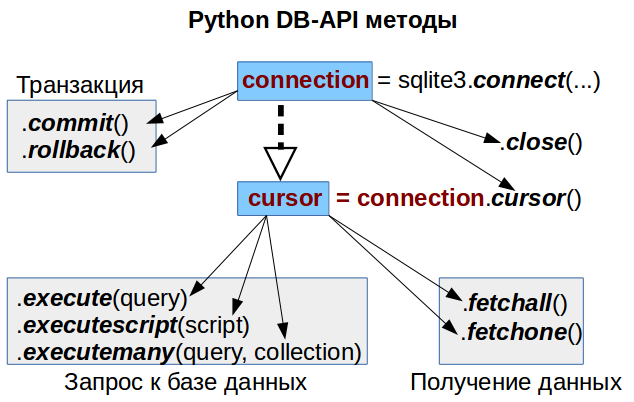
*Інструментарій Windows для роботи з Python*

В Windows робота з Python може бути виконана власними засобами Python – вікно чи оболонка IDLE, або за допомогою сторонніх IDE, наприклад PyCharm Community Edition.

*Інструментарій Python для роботи з БД*

Для роботи Python з БД використовується набір правил та інтерфейсів Python DB-API. Окремі деталі реалізації для різних БД можуть відрізнятися, але загальні принципи дозволяють використовувати один и той же підхід при роботі з різними БД.

На рисунку нижче наведено основні інтерфейси та їх використання з СУБД sqlite3. Аналогічна схема має місце для PostgreSQL та MySQL. Для доступу до інших баз використовуються звернення до ODBC - стандартного DB-API Windows для роботи з СУБД всіх виробників.



Для роботи з конкретною БД потрібно інсталювати пакети або модулі доступу до відповідної БД. Модулі DB-API в залежності від БД наведені в таблиці

|  |  |
| --- | --- |
| **База даних** | **DB-API модуль** |
| SQLite | sqlite3 |
| PostgreSQL | psycopg2 |
| MySQL | mysql.connector |
| ODBC | pyodbc |

Таким чином для роботи Python з PostgreSQL потрібно інсталювати модуль (пакет) доступу psycopg2.

Якщо для роботи з Python використовується вікно Python чи оболонка IDLE, пакет psycopg2 має бути встановлено в командному рядку Windows.

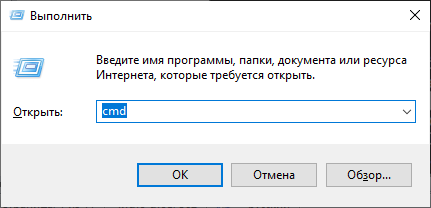
Якщо для роботи з Python використовується IDE, наприклад PyCharm Community Edition, то пакет psycopg2 потрібно інсталювати засобами цього IDE.

В будь якому випадку інстальований пакет роботи з БД, наприклад модуль psycopg2, треба імпортувати в прикладну програму роботи з БД.

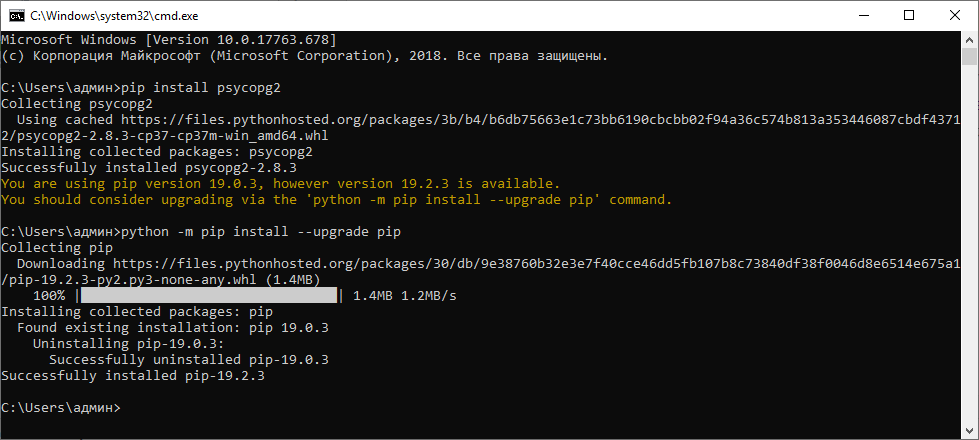
**Робота з БД власними засобами Python**

*Інсталяція модуля psycopg2 в командному рядку* *Windows*

Викликаємо командний рядок: Пуск/Виконати команда cmd.



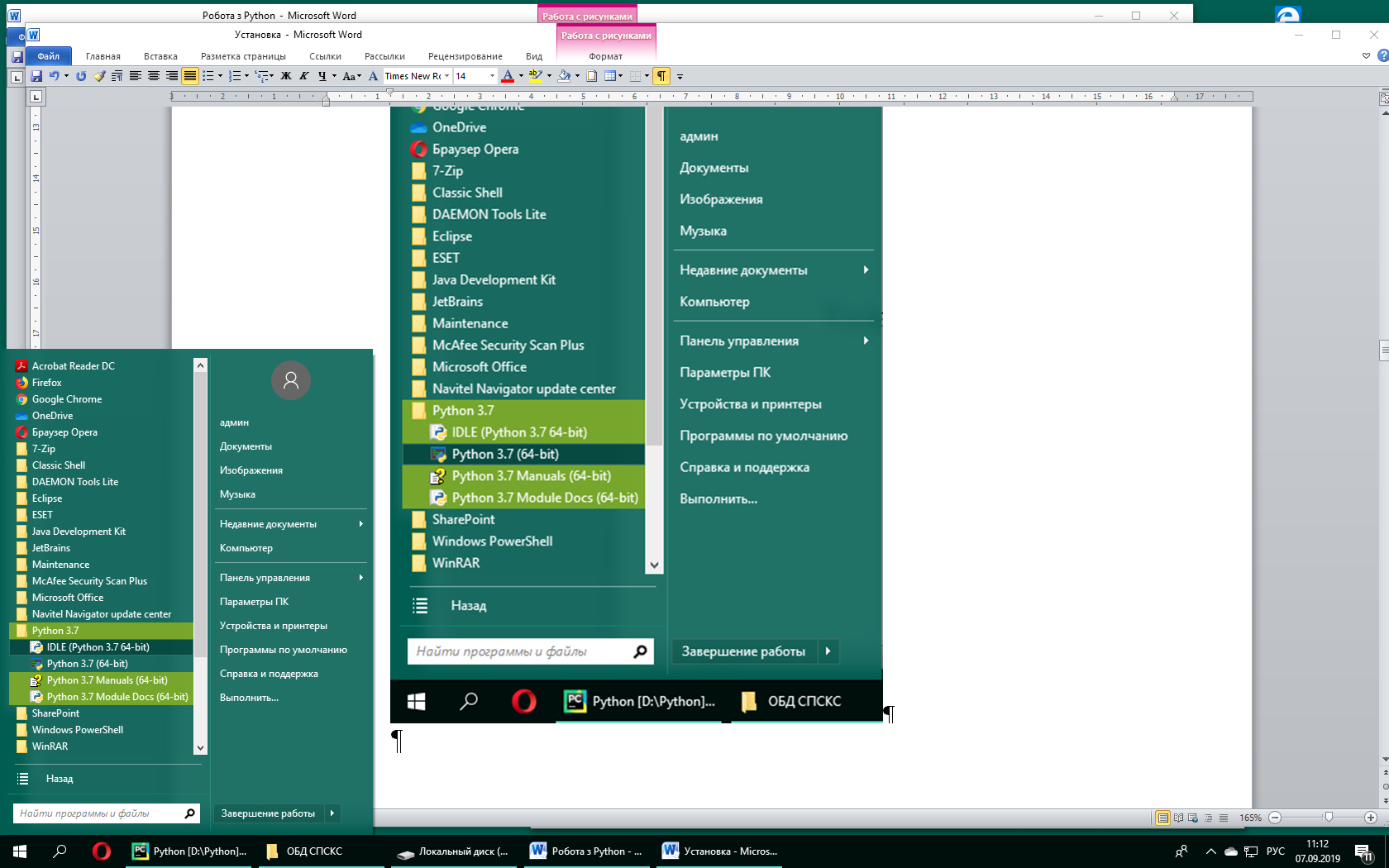
В командному вікні Windows виконуємо pip install psycopg2, де pip (Python Package Installer) інсталятор пакетів Python



Попутно можемо отримати повідомлення про наявність більш нової версії pip, тоді встановлюємо її.

*Виклик оболонки IDLE або вікна Python*

Для виклику оболонки IDLE або вікна Python виконуємо Пуск/Python 3.7 і далі за потребою

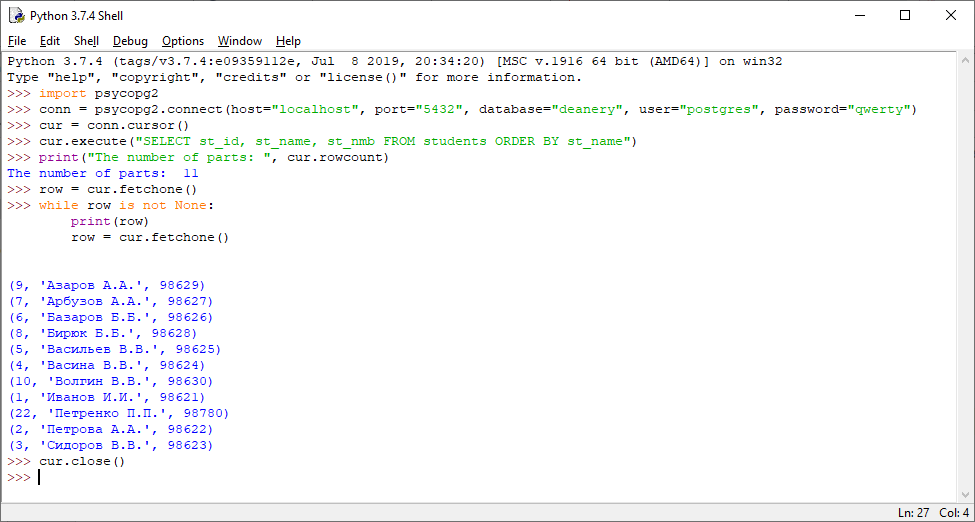


Для роботи з БД треба імпортувати в програму модуль psycopg2

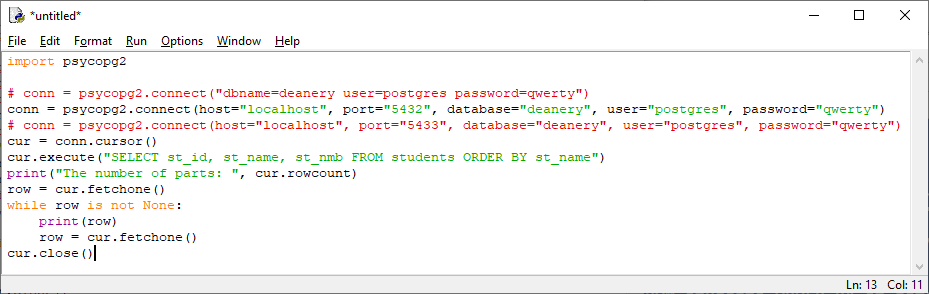
Доступ до БД в оболонці IDLE

Послідовно набираємо і виконуємо в оболонці Shell оператори Python.

*Увага!* Першим має бути оператор імпорту в проект модуля роботи з БД psycopg2

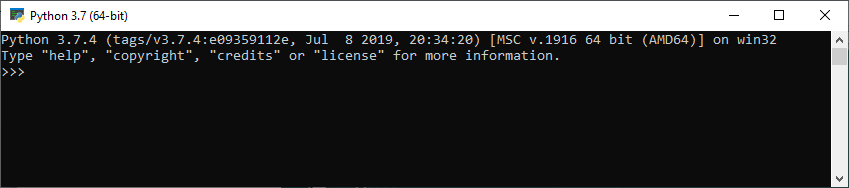


Можна завантажити в IDLE файл с підготовленим раніше текстом програми в форматі .txt, після чого зберегти його в форматі .py і виконати запит через пункт меню Run (F5)



Доступ до БД у вікні Python

Аналогічно доступу до БД в оболонці IDLE можна виконати доступ до БД у вікні Python 3.7, послідовно набираючи і виконуючи в оболонці оператори Python

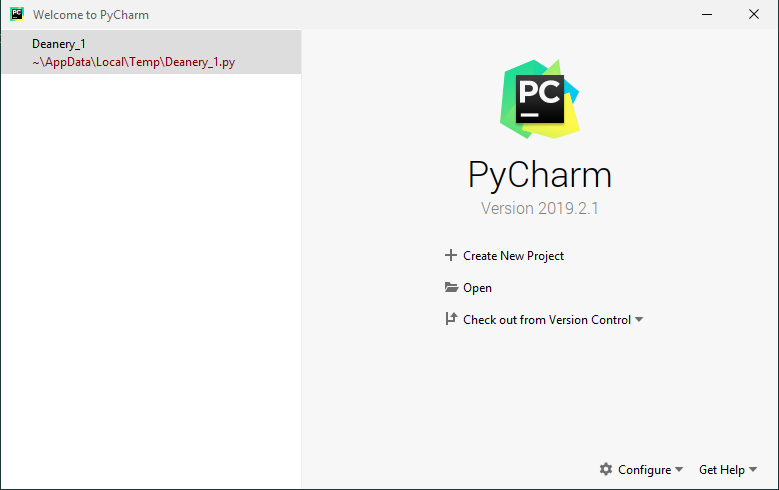


*Увага!* Першим має бути оператор імпорту в проект модуля роботи з БД psycopg2

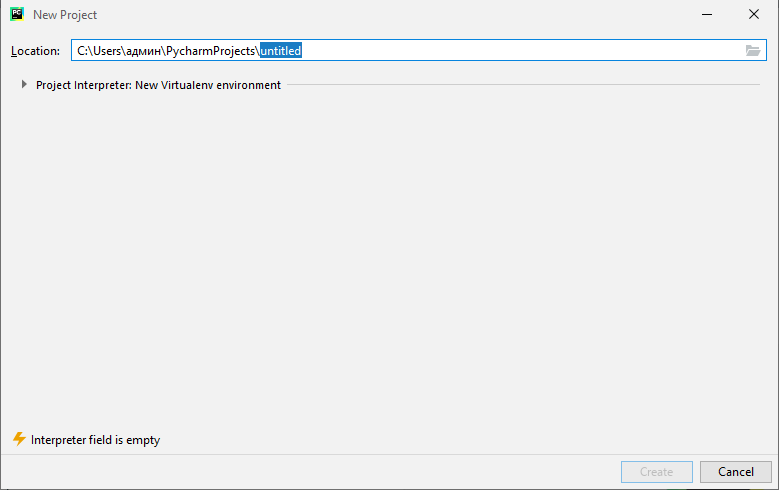
**Робота з БД засобами IDE PyCharm**

Створення проекту з БД в IDE PyCharm

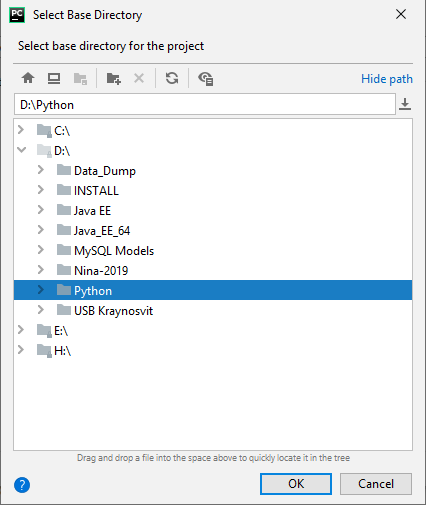
Створення першого проекту в PyCharm проводиться наступним чином



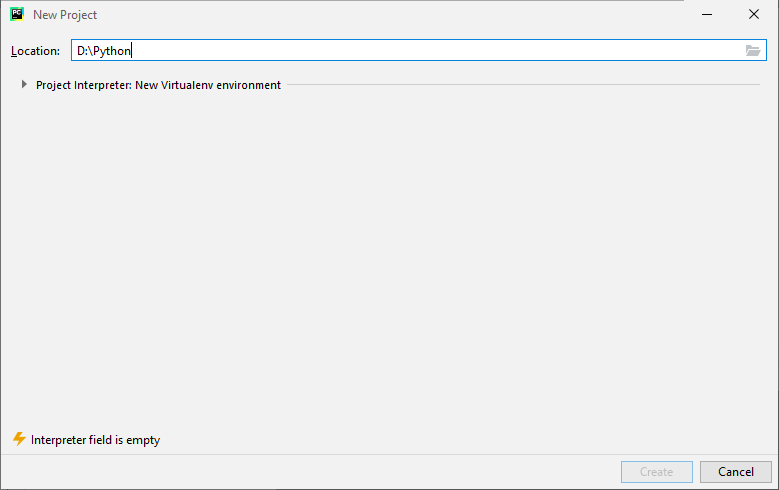
Програма запропонує місце розташування проекту. Можна з ним погодитися, але краще створити власний каталог, наприклад, на диску D:



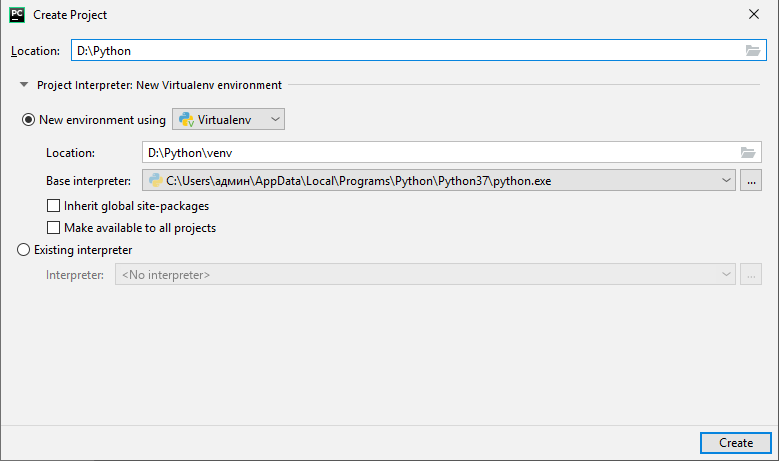
Вибираємо власний каталог проекту D:\Python



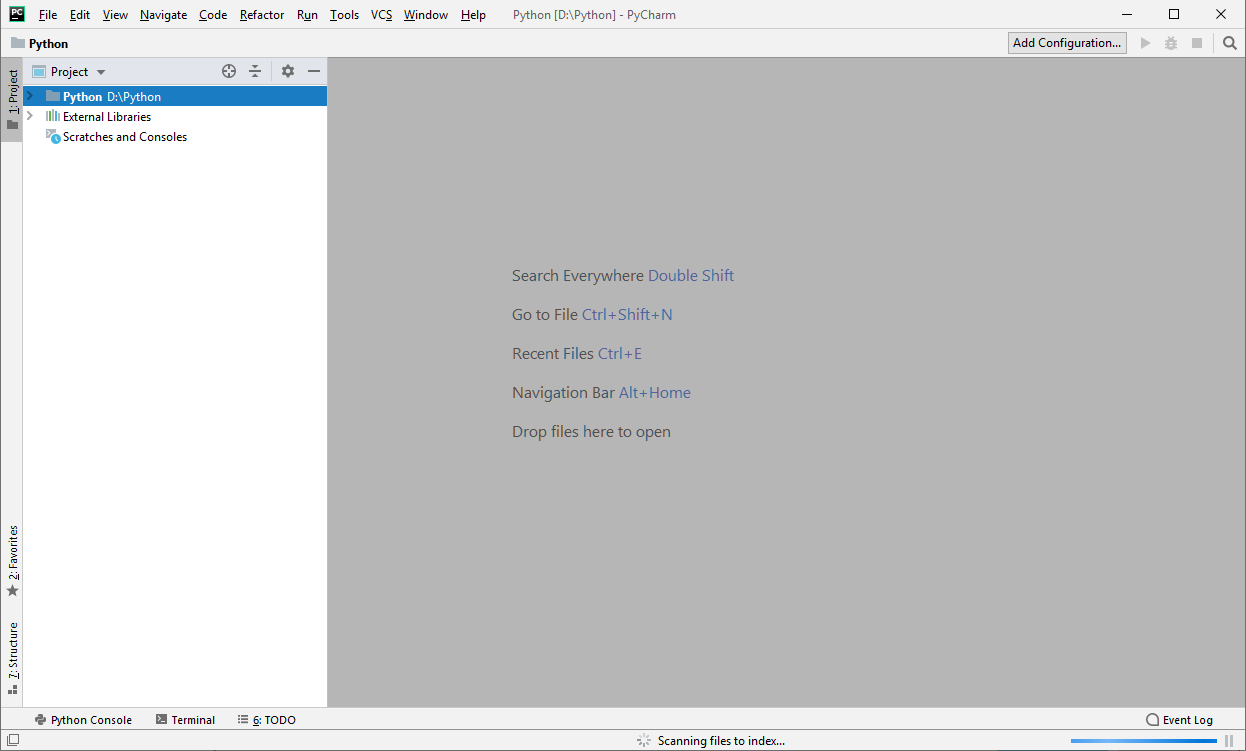
Далі можна побачити налаштування нового віртуального середовища інтерпретатора проекту. Якщо потрібно уточнюємо базовий інтерпретатор



Параметри налаштування проекту

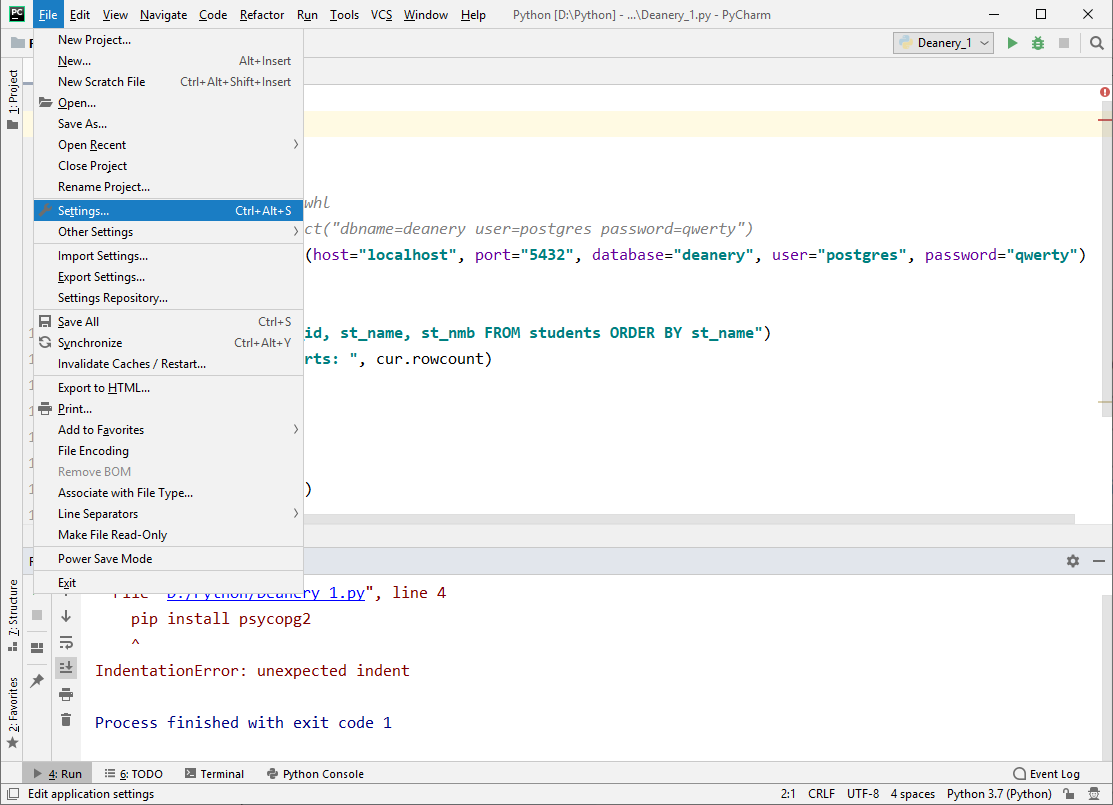


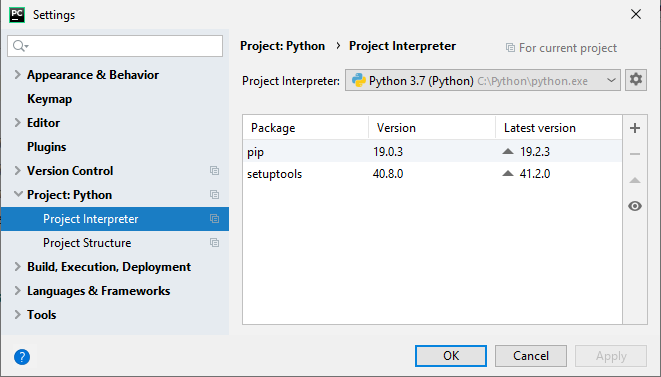
Тепер можна створити власний проект

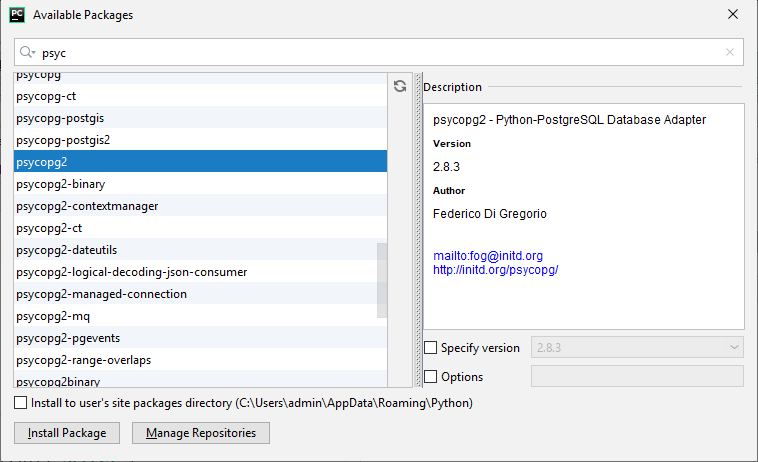


Створення проекту з БД в PyCharm

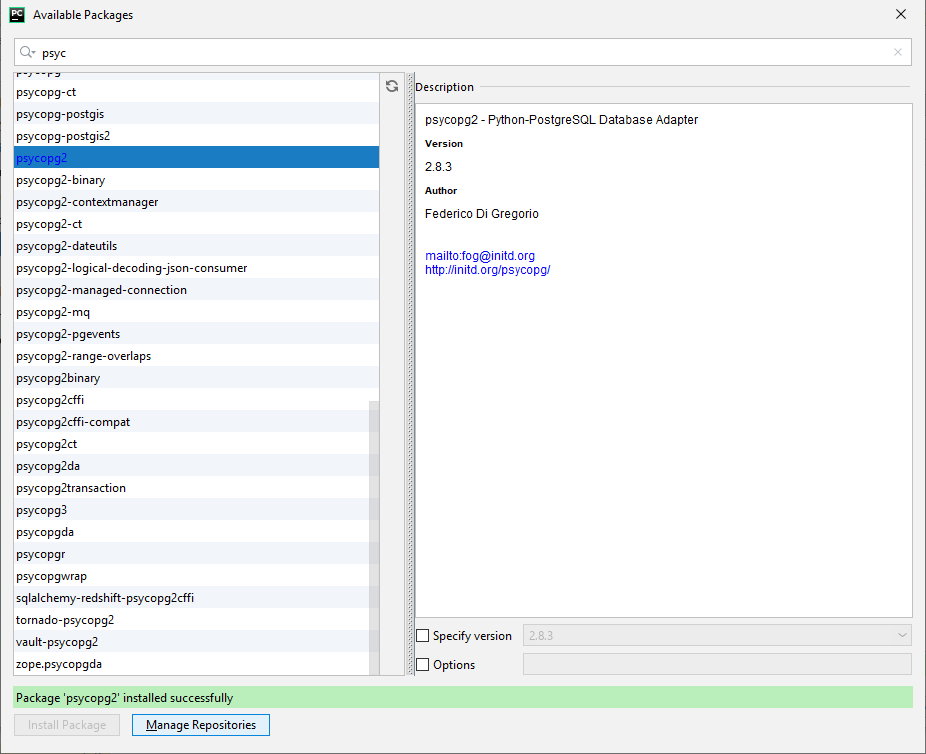
Для роботи з БД в PyCharm треба спочатку інсталювати модуль psycopg2. Для цього виконуємо



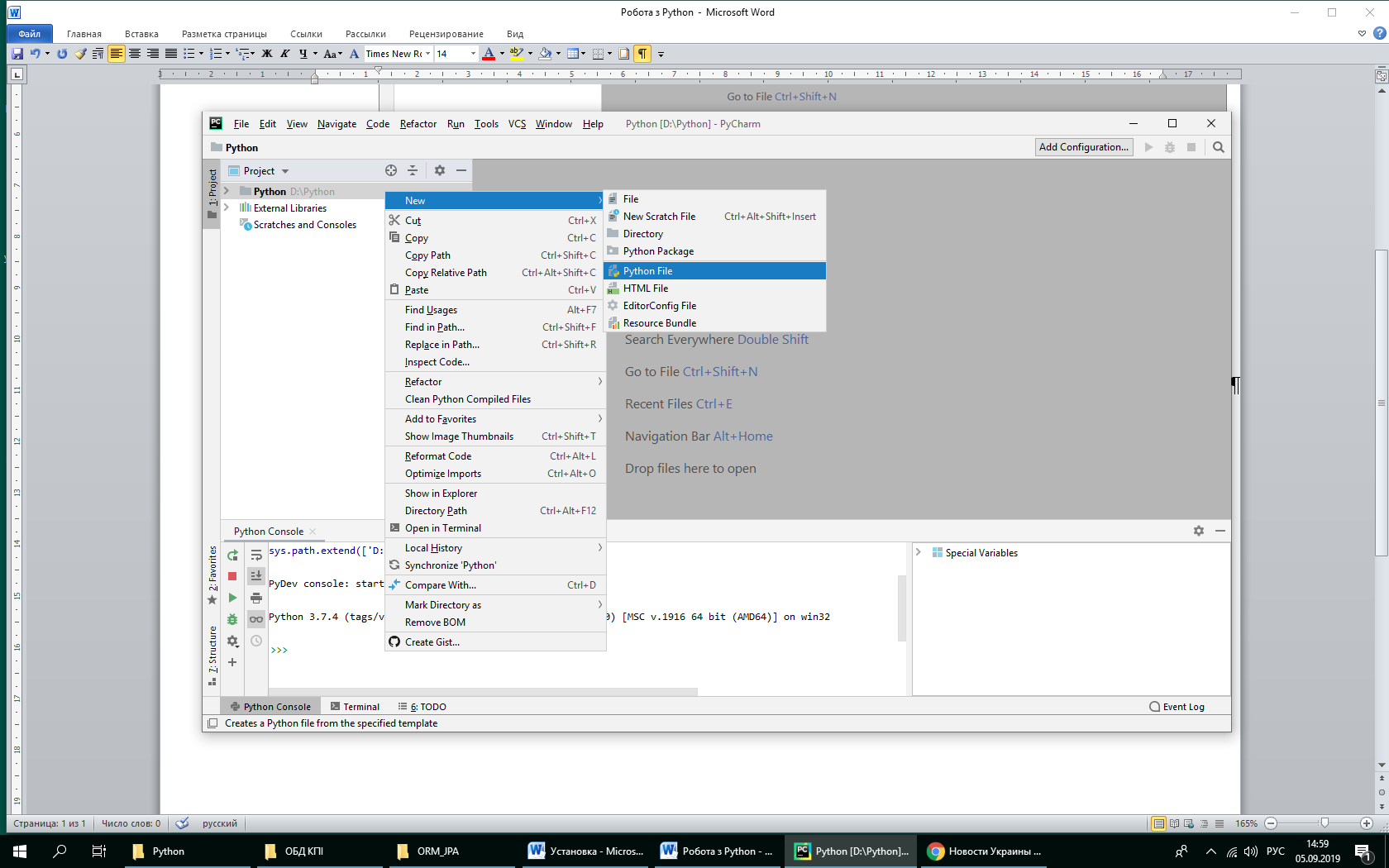




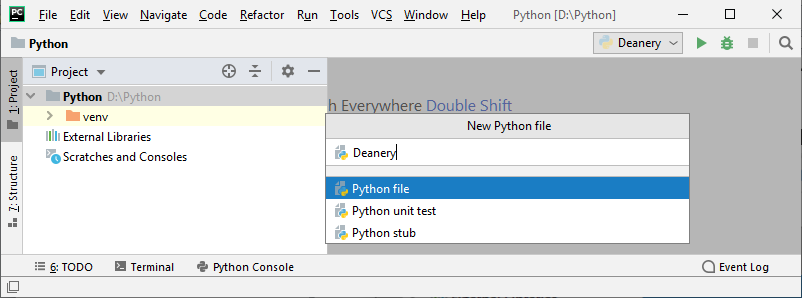
В разі успіху внизу вікна з’явиться повідомлення



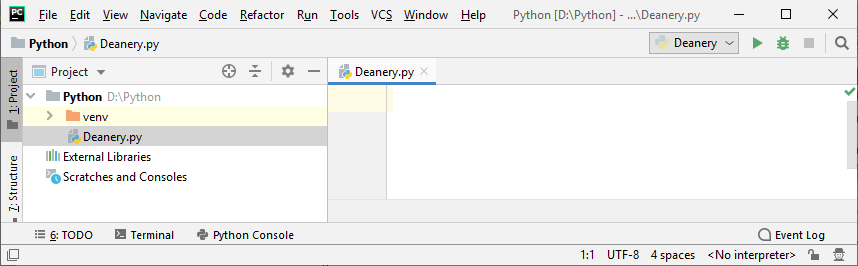
Тепер треба створити файл Python



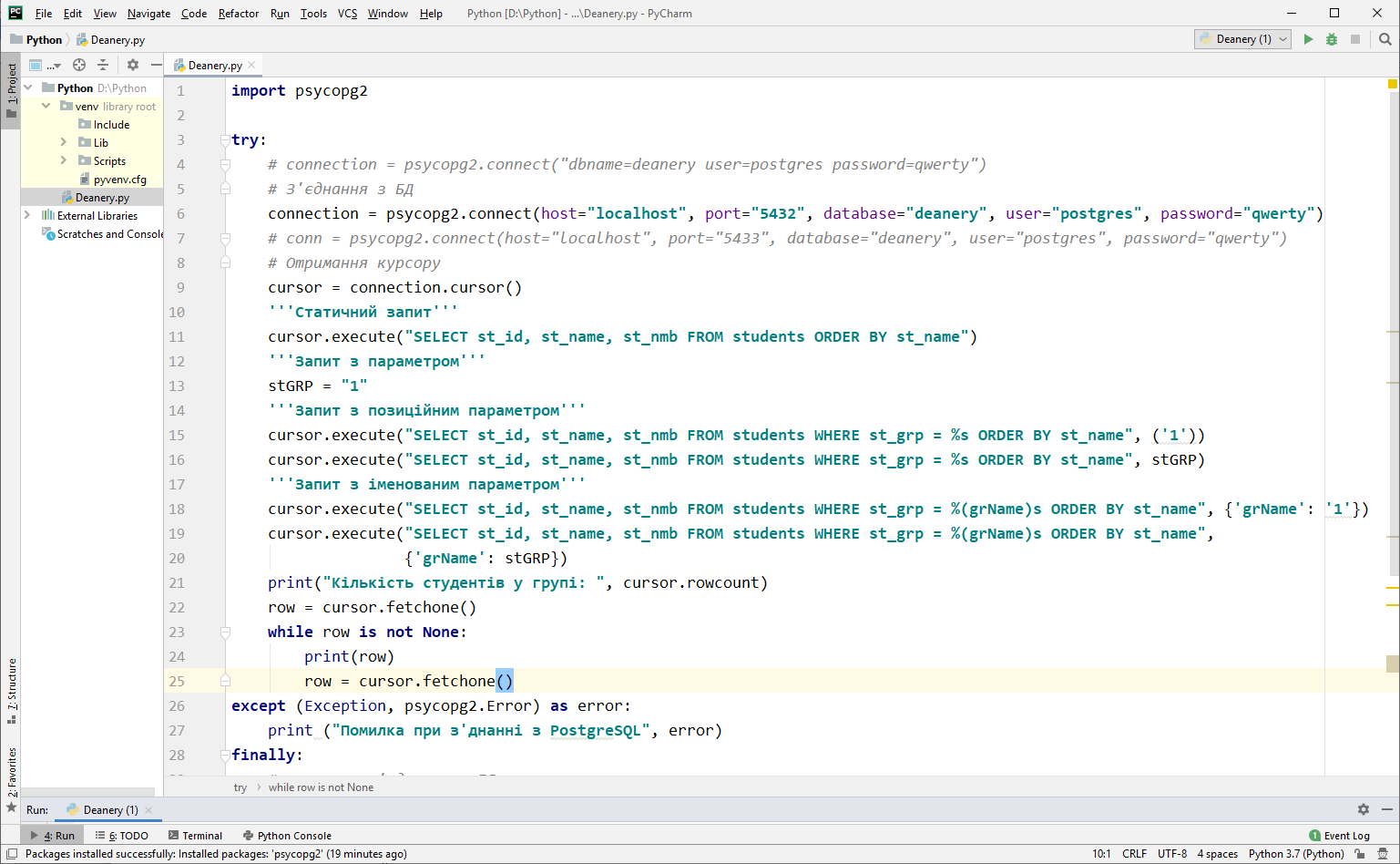
Задаємо ім’я файлу, наприклад Deanery



Отримаємо



Приклад файлу програми для роботи з БД наведено нижче. Слід підкреслити, що для роботи з БД в програму треба імпортувати модуль psycopg2



Тепер можна перейти до розробки конкретного проекту

**Робота з БД PostgreSQL засобами Python**

Для роботи з БД PostgreSQL треба виконати наступні дії:

1. Імпортувати пакет psycopg2;

|  |
| --- |
| **import** psycopg2 |

1. З’єднатися з БД;

|  |
| --- |
| connection = psycopg2.connect(host=**" localhost"**, port=**"5432"**,  database=**"deanery"**, user=**"postgres"**, password=**"qwerty"**) |

1. Отримати курсор. Через курсор проходить подальше спілкування з БД.

|  |
| --- |
| cursor = connection.cursor() |

1. Створити та обробити результати запиту

|  |
| --- |
| cursor.execute(**"SELECT st\_id, st\_name, st\_nmb FROM students**  **ORDER BY st\_name"**)  *# отримання результату запиту*  row = cursor.fetchone()  *# обробка результату запиту*  **while** row **is not** None:  print(row)  **...**  row = cursor.fetchone()  **...** |

Після виконання запиту, отримати результат можна декількома способами:

1. cursor – зразу забирає всі дані з серверу і повертає ітерабельний об’єкт
2. row = cursor.fetchone() - повертає 1-н рядок, або None, якщо нема більше даних;
3. row = cursor.fetchall() - повертає список всіх рядків або пустий список, якщо нема більше даних;
4. row = cursor.fetchmany(size) - повертає список з заданою кількістю рядків або пустий список, якщо нема більше даних.

Стандартний курсор забирає всі дані з сервера зразу, незалежно від того, використовуємо ми fetchall() або fetchone()

У випадку а, коли курсор повертає ітерабельний об’єкт, для обробки всіх повернутих рядків використовується цикл for …

|  |
| --- |
| **for** row **in** cursor:  print (row)  **…**  **...** |

У випадку b, коли курсор повертає 1-н рядок - fetchone(), для обробки всіх повернутих рядків використовується цикл while …

|  |
| --- |
| row = cursor.fetchone()  **while** row **is not** None:  print(row)  **...**  row = cursor.fetchone()  **...** |

У випадку с, коли курсор повертає всі рядки - fetchall(), для обробки всіх повернутих рядків використовується цикл for … in…

|  |
| --- |
| rows = cursor.fetchall()  **for** row **in** rows:  print (row)  **…**  **…** |

У випадку d, курсор повертає кількість рядків - fetchmany(число), задану у аргументі. Якщо цей метод викликати знов, він повертає наступний набір рядків результату запиту. Ці виклики можна робити послідовно або в циклі while. Якщо більше рядків нема, курсор повертає пустий список. Для обробки всіх повернутих рядків по аналогії з fetchall() використовується цикл for … in ….

|  |
| --- |
| **while** True:  rows = cursor.fetchmany(5)  **if** **not** rows:  **break**  **for** row **in** rows:  print (row)  **…** |

1. Хорошою практикою при роботі з БД є закриття курсору та з’єднання з БД.

|  |
| --- |
| cursor.close()  connection.close() |

1. Робота з БД пов’язана з великою кількістю можливих проблем – нема зв’язку з сервером БД, не працює сервер БД, на сервері нема відповідної БД і т.п.

Для того, щоб програма не завершилася аварійно треба передбачити контрольоване завершення невдалого звернення до БД. Для цього використовується конструкція try … except … finally.

|  |
| --- |
| **try**:  *# Звернення до БД*  **…**  **except** (Exception, psycopg2.Error) **as** error:  print (**"Помилка при з'єднанні з PostgreSQL"**, error)  **finally**:  *# Закриття з'єднання з БД.*  **if** (connection):  cursor.close()  connection.close()  print(**"З'єднання з PostgreSQL закрите"**) |

Особливості обробки результатів запиту

Рядки або кортежі

За замовчанням результат запиту до БД приходить у вигляді рядка (кортежу). В попередніх прикладах ці кортежі в повному обсязі виводилися на друк. Але часто в запиті користувача цікавлять окремі поля кортежу. Доступ до полів кортежу виконується по індексу. Індекси приймають значення, починаючи з "0". В прикладі нижче будуть виводитися ім’я студента та номер його студентського білету

|  |
| --- |
| cursor.execute(**"SELECT st\_id, st\_name, st\_nmb FROM students**  **ORDER BY st\_name"**)  *# отримання результату запиту*  row = cursor.fetchone()  *# обробка результату запиту*  **while** row **is not** None:  print(row[1], row[2])  **...**  row = cursor.fetchone()  **...** |

Якщо потрібно працювати зі словником, то при виклику cursor треба передати аргумент cursor\_factory.

Курсор-словник

Курсори-словники дозволяють отримати доступ до вибраних записів, використовуючи інтерфейс, подібний словникам Python, замість кортежів. В результаті до полів вибраних рядків можна звертатися по іменам. Для цього треба послідовно імпортувати psycopg2, extras та DictCursor і в оголошення курсору додати параметр cursor\_factory=DictCursor. При цьому записи підтримують індексацію, тобто ведуть себе як рядки, що дозволяє звернення до полів рядка по індексам:

|  |
| --- |
| **import** psycopg2  **from** psycopg2 **import** extras  **from** psycopg2.extras **import** DictCursor  connection = psycopg2.connect(host=**"localhost"**, port=**"5432"**,  database=**"deanery"**, user=**"postgres"**, password=**"qwerty"**)  cursor = connection.cursor(cursor\_factory=DictCursor)  cursor.execute(**"SELECT st\_id, st\_name, st\_nmb FROM students**  **ORDER BY st\_name"**)  row = cursor.fetchone()  **while** row **is not None**:  print(row)  print(row[1], row[2])  print(row[**'st\_name'**], row[**'st\_nmb'**])  row = cursor.fetchone() |

Параметричні запити. Підстановка значень в запиті

Параметричні запити використовуються, коли значення параметрів в запитах заздалегідь невідомі і формується динамічно під час виконання програми. Такі параметричні параметри запиту формуються за допомогою другого параметру метода execute().

Метод execute() другим параметром приймає колекцію (кортеж, список и т.д.) або словник. При формуванні запиту необхідно пам’ятати, що:

* плейсхолдеры в рядку запиту мають бути %s, навіть якщо тип передаваємого значення відрізняється від рядка, всю роботу бере на себе psycopg2;
* не треба брати рядки в одинарні лапки;
* якщо в запиті присутній знак %, то його необхідно писати як пару знаків %%.

Параметри в запиті можуть бути *позиційні* та *іменовані*.

Позиційні параметри формуються за допомогою конструкції %s.

Іменовані параметри формуються за допомогою конструкції %(ім’я\_параметра)s.

Якщо позиційних параметрів декілька, то порядок підстановки значень цих параметрів є суттєвим і виконується зліва на право.

|  |
| --- |
| **'''Запит з позиційним параметром'''**  cursor.execute(**"SELECT st\_id, st\_name, st\_nmb**  **FROM students WHERE st\_grp = %s**  **ORDER BY st\_name"**, (**'КВ-71'**)  *# або так*  stGRP = **"КВ-71"**  cursor.execute(**"SELECT st\_id, st\_name, st\_nmb FROM students WHERE st\_grp = %s ORDER BY st\_name"**, stGRP) |

Якщо іменованих параметрів декілька, то порядок підстановки їм значень несуттєвий.

|  |
| --- |
| **'''Запит з іменованим параметром'''**  cursor.execute**("SELECT st\_id, st\_name, st\_nmb**  **FROM students WHERE st\_grp = %(grName)s**  **ORDER BY st\_name"**, {**'grName': 'КВ-71'**}  *# або так*  stGRP = **"КВ-71"**  cursor.execute**("SELECT st\_id, st\_name, st\_nmb**  **FROM students WHERE st\_grp = %(grName)s**  **ORDER BY st\_name"**, {**'grName':** stGRP) |

**Операції додавання, модифікації та видалення записів**

Зазвичай операції додавання, модифікації та видалення записів реалізуються параметризованими SQL-запитами

Вставка нових записів. Директива INSERT

|  |
| --- |
| **try**:  connection = psycopg2.connect(host=**"localhost"**, port=**"5432"**,  database=**"deanery"**, user=**"postgres"**, password=**"qwerty"**)  cursor = connection.cursor()  stName = **'Майстренко В.В.'**  stNmb = 98789  cursor.execute(**"INSERT INTO students (st\_name, st\_nmb)**  **VALUES (%s, %s)"**, (stName, stNmb))  connection.commit()  **except** (Exception, psycopg2.Error) **as** error:  print (**"Помилка при з'єднанні з PostgreSQL"**, error)  **finally**:  **if** (connection):  cursor.close()  connection.close() |

Оновлення існуючих записів. Директива UPDATE

|  |
| --- |
| **try**:  connection = psycopg2.connect(host=**"localhost"**, port=**"5432"**,  database=**"deanery"**, user=**"postgres"**, password=**"qwerty"**)  cursor = connection.cursor()  stName = **'Шолуденко В.В.'**  stNmb = 98789  cursor.execute(**"UPDATE students SET st\_name = %s**  **WHERE st\_nmb = %s"**, (stName, stNmb))  connection.commit()  **except** (Exception, psycopg2.Error) **as** error:  print (**"Помилка при з'єднанні з PostgreSQL"**, error)  **finally**:  **if** (connection):  cursor.close()  connection.close() |

Видалення існуючих записів. Директива DELETE

|  |
| --- |
| **try**:  connection = psycopg2.connect(host=**"localhost"**, port=**"5432"**,  database=**"deanery"**, user=**"postgres"**, password=**"qwerty"**)  cursor = connection.cursor()  stNmb = 98789  cursor.execute(**"DELETE FROM students**  **WHERE st\_nmb = %s"**, (stNmb))  connection.commit()  **except** (Exception, psycopg2.Error) **as** error:  print (**"Помилка при з'єднанні з PostgreSQL"**, error)  **finally**:  **if** (connection):  cursor.close()  connection.close() |

Операції по вставці, оновленню та видаленню груп записів

Python PostgreSQL дозволяє виконувати операції по вставці, оновленню та видаленню груп записів. Для цього спочатку готується відповідний до типу операції набір записів. Після чого використовується директива виду

cursor.executemany(запит, набір записів)

Покажемо це на прикладі вставки нових записів. Директива INSERT

|  |
| --- |
| **try**:  connection = psycopg2.connect(host=**"localhost"**, port=**"5432"**,  database=**"deanery"**, user=**"postgres"**, password=**"qwerty"**)  cursor = connection.cursor()  recordsToIns = [ (**'Майстренко В.В.'**, 98789),  (**'Філоненко М.М.'**, 98791) ]  cursor.executemany(**"INSERT INTO students (st\_name, st\_nmb)**  **VALUES (%s, %s)"**, recordsToIns)  connection.commit()  **except** (Exception, psycopg2.Error) **as** error:  print (**"Помилка при з'єднанні з PostgreSQL"**, error)  **finally**:  **if** (connection):  cursor.close()  connection.close() |

Література

Python: Работа с базой данных, часть 1/2: Используем DB-API

<https://habr.com/ru/post/321510/>

Python PostgreSQL Tutorial Using Psycopg2

<https://pynative.com/python-postgresql-tutorial/>

PostgreSQL Python: Querying Data

<http://www.postgresqltutorial.com/postgresql-python/query/>

Python: Работа с базой данных, часть 1/2: Используем DB-API

<https://habr.com/ru/post/321510/>

Работа с PostgreSQL в Python

<https://khashtamov.com/ru/postgresql-python-psycopg2/>

Python cursor’s fetchall, fetchmany(), fetchone() to read records from database table

<https://pynative.com/python-cursor-fetchall-fetchmany-fetchone-to-read-rows-from-table/>

Python PostgreSQL Insert, Update and Delete table data to Perform CRUD Operations

<https://pynative.com/python-postgresql-insert-update-delete-table-data-to-perform-crud-operations/>