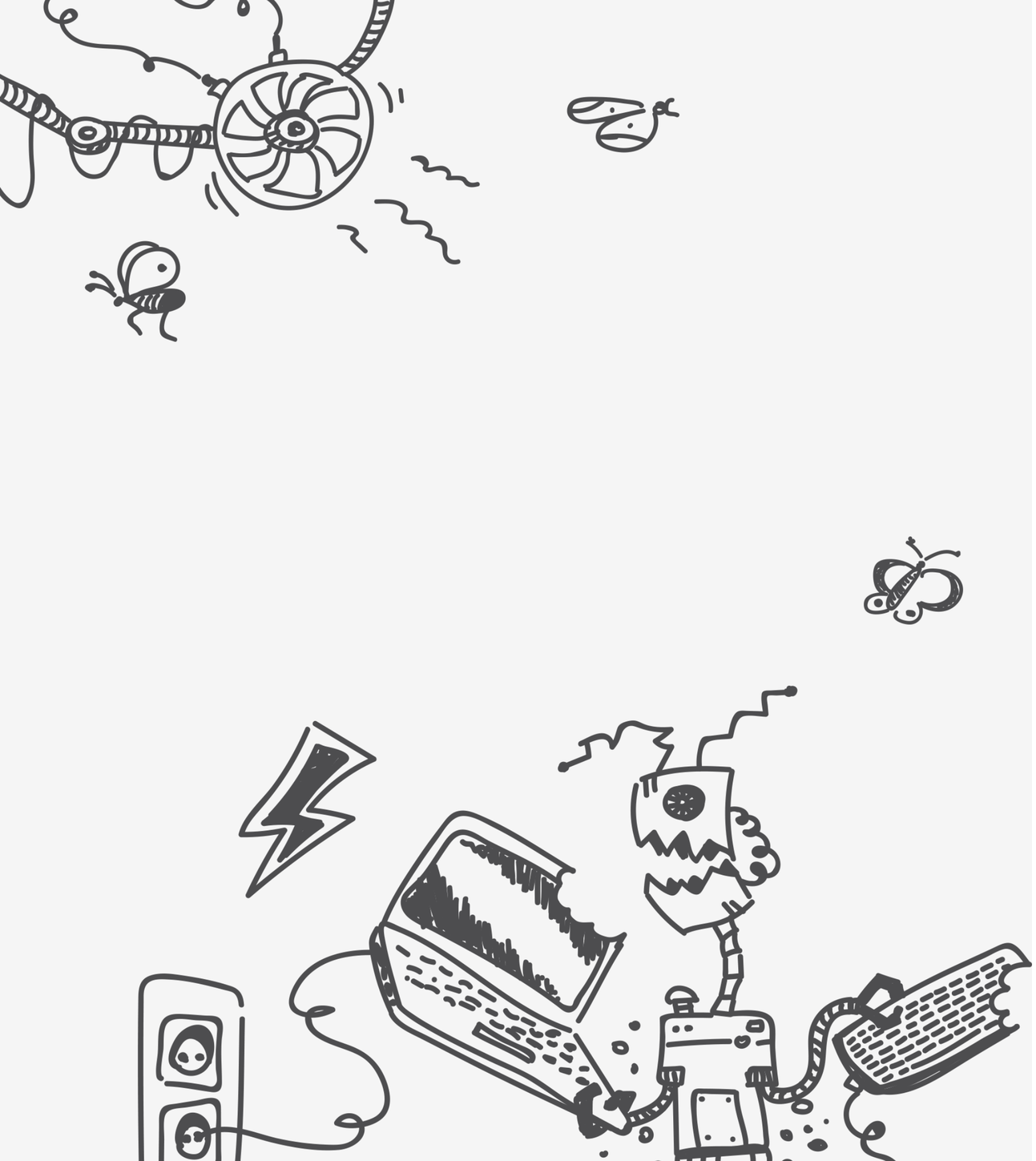

PostgreSQL i Python

Interfejs PostgreSQL w Pythonie

Najpopularniejszym ze sterowników PostgreSQL dla języka Python jest **psycopg2**.

Aby go zainstalować, najlepiej jest stworzyć wirtualne środowisko (używając konsoli) komendą `virtualenv`:

```
virtualenv -p python3 env
```

A następnie je aktywować:

```
source env/bin/activate
```

Gdy już mamy aktywowane środowisko, użyjmy programu `pip`, który zainstaluje nam odpowiednią bibliotekę:

```
pip install psycopg2-binary
```

lub (jeśli powyższe polecenie nie zadziała)

```
pip3 install psycopg2-binary
```

Więcej informacji znajdziesz tutaj: http://initd.org/psycopg/docs/

Praca z bazą danych

Łączenie się z bazą

Pierwszym krokiem do używania bazy danych jest podłączenie się do niej. Aby połączyć się z bazą danych, należy utworzyć połączenie. Połączenie to obiekt klasy `connection`, który tworzymy następująco:

```
from psycopg2 import connect
cnx = connect(user=<login>,
              password=<hasło>,
              host=<ip-serwera>,
              database=<nazwa-bazy>)
```

Jeżeli podamy nazwę bazy, od razu się do niej podłączymy.

Kończenie pracy z bazą

Musimy też pamiętać o poprawnym zamknięciu połączenia.

```
cnx.close()
```

http://initd.org/psycopg/docs/connection.html

Sprawdzanie błędów

Jeśli operacja zakończy się błędem, to Python o tym poinformuje, rzucając wyjątek. Na pewno go zauważycie: ;-)

```
cnx = connect(user="root", password="bad-pass", host="127.0.0.1")
```

```
File "pg.py", line 8, in <module>
    password="bad-pass")
  File "/home/kaczor/workspace/test-python/env/lib/python3.6/site-packages/psycopg2/__init__.
    conn = _connect(dsn, connection_factory=connection_factory, **kwasync)
psycopg2.OperationalError: FATAL:  password authentication failed for user "postgres"
FATAL:  password authentication failed for user "postgres"
```

Sprawdzanie błędów

Jeśli operacja zakończy się błędem, to Python o tym poinformuje, rzucając wyjątek. Na pewno go zauważycie: ;-)

```
cnx = connect(user="root", password="bad-pass", host="127.0.0.1")
```

```
File "pg.py", line 8, in <module>
    password="bad-pass")
  File "/home/kaczor/workspace/test-python/env/lib/python3.6/site-packages/psycopg2/__init__.
    conn = _connect(dsn, connection_factory=connection_factory, **kwasync)
psycopg2.OperationalError: FATAL:  password authentication failed for user "postgres"
FATAL:  password authentication failed for user "postgres"
```

→ **`password="bad-pass"`** - W przykładzie użyliśmy błędnego hasła

Przykład połączenia

```
from psycopg2 import connect, OperationalError
username = "postgres"
passwd = "coderslab"
hostname = "127.0.0.1"  # lub "localhost"
db_name = "my_first_db"
try:
    # tworzymy nowe połączenie
    cnx = connect(user=username, password=passwd, host=hostname, database=db_name)
    print("Połączenie udane.")
    cnx.close()
except OperationalError:
    print("Nieudane połączenie.")
```

Praca z połączeniem

W Pythonie komunikacja z PostgreSQL polega na wysyłaniu zapytań (`queries`) przez obiekt klasy `cursor`. Obiekt ten tworzony jest podczas połączenia z bazą danych. Można go uzyskać przez odpytanie obiektu połączenia:

```
cursor = cnx.cursor()
```

Następnie można zadać pytanie do bazy przez wywołanie metody `execute` z zapytaniem SQL jako parametrem:

```
cursor.execute(<zapytanie sql>)
```

Transakcje

Zazwyczaj chcemy wykonywać operacje SQL jedna po drugiej. Czasem jednak zdarza się, że potrzebujemy wykonać kilka operacji naraz. Przykładowo, jeśli wykonujemy przelew w banku, to chcemy, aby stan jednego konta zmalał, a innego wzrósł. Aby zagwarantować wykonanie obu operacji naraz, używamy transakcji.

W `psycopg2` domyślnie włączony jest tryb transakcji. Oznacza to, że samo wywołanie komendy `cursor.execute(<zapytanie sql>)` nie spowoduje żadnej zmiany w bazie danych. Musimy wywołać potem metodę `commit()` na obiekcie połączenia (`cnx.commit()`), aby zatwierdzić transakcję i wykonać wszystkie zapytania.

Transakcje

Domyślnie musimy zatwierdzić zapytanie wywołując metodę `commit()`.

```
...
cursor.execute(sql)
cnx.commit()
```

Możemy wyłączyć transakcje poprzez ustawienie zmiennej `connection.autocommit` na `True`.

```
...
cnx.autocommit = True
cursor.execute(sql)
```

Transakcje

Domyślnie musimy zatwierdzić zapytanie wywołując metodę `commit()`.

```
...
cursor.execute(sql)
cnx.commit()
```

→ Uruchamiamy zapytane sql używając wcześniej utworzonego obiektu kursora (`cursor`).

Możemy wyłączyć transakcje poprzez ustawienie zmiennej `connection.autocommit` na `True`.

```
...
cnx.autocommit = True
cursor.execute(sql)
```

Transakcje

Domyślnie musimy zatwierdzić zapytanie wywołując metodę `commit()`.

```
...
cursor.execute(sql)
cnx.commit()
```

→ Uruchamiamy zapytane sql używając wcześniej utworzonego obiektu kursora (`cursor`).

W tym momencie zapytanie zostaje wywołane na bazie danych.

Możemy wyłączyć transakcje poprzez ustawienie zmiennej `connection.autocommit` na `True`.

```
...
cnx.autocommit = True
cursor.execute(sql)
```

Transakcje

Domyślnie musimy zatwierdzić zapytanie wywołując metodę `commit()`.

```
...
cursor.execute(sql)
cnx.commit()
```

→ Uruchamiamy zapytane sql używając wcześniej utworzonego obiektu kursora (`cursor`).

W tym momencie zapytanie zostaje wywołane na bazie danych.

Możemy wyłączyć transakcje poprzez ustawienie zmiennej `connection.autocommit` na `True`.

```
...
cnx.autocommit = True
cursor.execute(sql)
```

Wyłączamy transakcje.

Transakcje

Domyślnie musimy zatwierdzić zapytanie wywołując metodę `commit()`.

```
...
cursor.execute(sql)
cnx.commit()
```

- Uruchamiamy zapytane sql używając wcześniej utworzonego obiektu kursora (`cursor`).
- W tym momencie zapytanie zostaje wywołane na bazie danych.

Możemy wyłączyć transakcje poprzez ustawienie zmiennej `connection.autocommit` na `True`.

```
...
cnx.autocommit = True
cursor.execute(sql)
```

Wyłączamy transakcje.

Zapytanie od razu jest wywoływane na bazie danych.

Context manager

Możemy korzystać z transakcji bez konieczności wywoływania `commit()`.

```
conn = psycopg2.connect(...)
with conn.cursor() as curs:
    curs.execute(SQL1)

with conn.cursor() as curs:
    curs.execute(SQL2)

conn.close()
```

Składnia `with something as some_name:` to tzw. context manager - konstrukcja która dba o to, aby:

- przed blokiem kodu wykonać jakąś operację (tutaj: rozpocząć transakcję),
- po poprawnym opuszczeniu bloku kodu wykonać jakąś operację (tutaj: zakończyć transakcję),
- jeśli blok kodu zakończył się wyjątkiem, wykonać jakąś inną operację (tutaj: anulować transakcję)

Tworzenie nowej bazy danych przez Pythona

```
from psycopg2 import connect, OperationalError
sql = "CREATE DATABASE sql_cwiczenia;"
try:
    cnx = connect(user="postgres", password="coderslab", host="127.0.0.1")
    cnx.autocommit = True
    cursor = cnx.cursor()
    cursor.execute(sql)
    print("Baza założona")
except OperationalError:
    print("Błąd!")
else:
    cursor.close()
    cnx.close()
```

Tworzenie nowej bazy danych przez Pythona

```
from psycopg2 import connect, OperationalError
sql = "CREATE DATABASE sql_cwiczenia;"
try:
    cnx = connect(user="postgres", password="coderslab", host="127.0.0.1")
    cnx.autocommit = True
    cursor = cnx.cursor()
    cursor.execute(sql)
    print("Baza założona")
except OperationalError:
    print("Błąd!")
else:
    cursor.close()
    cnx.close()
```

→ **Nie podajemy parametru `database`, ponieważ będziemy tworzyć nową bazę.**

Tworzenie nowej bazy danych przez Pythona

```
from psycopg2 import connect, OperationalError
sql = "CREATE DATABASE sql_cwiczenia;"
try:
    cnx = connect(user="postgres", password="coderslab", host="127.0.0.1")
    cnx.autocommit = True
    cursor = cnx.cursor()
    cursor.execute(sql)
    print("Baza założona")
except OperationalError:
    print("Błąd!")
else:
    cursor.close()
    cnx.close()
```

→ **Nie podajemy parametru `database`, ponieważ będziemy tworzyć nową bazę.**

Ustawiamy `autocommit` na `True`, aby polecenia wykonywały się od razu.

Tworzenie nowej bazy danych przez Pythona

```
from psycopg2 import connect, OperationalError
sql = "CREATE DATABASE sql_cwiczenia;"
try:
    cnx = connect(user="postgres", password="coderslab", host="127.0.0.1")
    cnx.autocommit = True
    cursor = cnx.cursor()
    cursor.execute(sql)
    print("Baza założona")
except OperationalError:
    print("Błąd!")
else:
    cursor.close()
    cnx.close()
```

→ **Nie podajemy parametru `database`, ponieważ będziemy tworzyć nową bazę.**

Ustawiamy `autocommit` na `True`, aby polecenia wykonywały się od razu.

Zawsze pamiętaj o poprawnym zamknięciu i zniszczeniu kursora połączenia.

Tworzenie nowej tabeli przez Pythona

Sposób pierwszy: ręczne zatwierdzenie transakcji

Pierwszym sposobem jest ręczne zatwierdzenie każdej transakcji. Odbywa się to przez użycie metody `commit()`:

```
sql = """CREATE TABLE users (user_id serial, user_name varchar(255),
        user_email varchar(255) UNIQUE, PRIMARY KEY(user_id))"""
try:
    cnx = connect(user=username, password=passwd, host=hostname, database=db_name)
    cursor = cnx.cursor()
    cursor.execute(sql)
    cnx.commit()  # zatwierdzenie transakcji
except:
    # obsługa błędu
```

Metodę `commit()` należy wywołać za każdym razem, gdy chcemy zapisać dane do bazy.

Tworzenie nowej tabeli przez Pythona

Sposób drugi: automatyczne zatwierdzenie transakcji

Drugim sposobem jest automatyczne zatwierdzanie każdej transakcji. Odbywa się to przez ustawienie atrybutu `autocommit` w obiekcie połączenia na `True`:

```
sql = """CREATE TABLE users (user_id serial, user_name varchar(255),
          user_email varchar(255) UNIQUE, PRIMARY KEY(user_id))"""
try:
    cnx = connect(user=username, password=passwd, host=hostname, database=db_name)
    cnx.autocommit = True
    cursor = cnx.cursor()
    cursor.execute(sql)
except:
    # obsługa błędu
```

Atrybut `autocommit` należy ustawić po nawiązaniu połączenia z bazą danych. Po wykonaniu metody `execute()` dane będą zapisywane automatycznie.

Dodawanie elementów do tabeli przez Pythona

```
sql = """INSERT INTO users(user_name, user_email)
         VALUES('Wojtek', 'wojtek@gmail.com')"""
try:
    cnx = connect(user=username, password=passwd, host=hostname, database=db_name)
    cnx.autocommit = True
    cursor = cnx.cursor()
    cursor.execute(sql)
except:
    pass
```

Ostatni wstawiony/edytowany element

- Po każdej operacji wstawienia lub edycji możemy otrzymać dowolną wartość elementu, na którym pracowaliśmy.
- Aby to zrobić do zapytania SQL należy dodać `RETURNING nazwa_kolumny`.
- Np:

```
sql = """INSERT INTO users
      (user_name, user_email)
      VALUES('Wojtek',
             'wojtek@gmail.com')
      RETURNING id """
```

```
last_id = cursor.fetchone()[0]
print("Nowy rekord o id {} został "
      "dodany".format(last_id))
```

Ostatni wstawiony/edytowany element

- Po każdej operacji wstawienia lub edycji możemy otrzymać dowolną wartość elementu, na którym pracowaliśmy.
- Aby to zrobić do zapytania SQL należy dodać `RETURNING nazwa_kolumny`.
- Np:

```
sql = """INSERT INTO users
      (user_name, user_email)
      VALUES('Wojtek',
             'wojtek@gmail.com')
      RETURNING id """
```

```
last_id = cursor.fetchone()[0]
print("Nowy rekord o id {} został "
      "dodany".format(last_id))
```

Metoda `fetchone` zwraca krotkę (tuple), gdzie pod indeksem 0 znajduje się wartość, która miała zostać zwrócona (w przykładzie jest to id nowo utworzonego rekordu).

Obiekt typu cursor

Metoda `execute` w przypadku udanych zapytań `SELECT`, zwraca nam `iterator`.

Aby dostać się do danych zawróconych przez zapytanie, należy przeiterować się przez obiekt `cursor`:

```
for row in cursor:
    print(row)
```

Iterator zawiera w sobie zestaw krotek z danymi. Jeśli wynikiem zapytania są dane użytkowników (id i name), będzie to wyglądało następująco:

```
(1, 'Wojtek')
(2, 'Wojtek2')
(3, 'Paweł')
(4, 'Janusz')
```

Wczytywanie elementów (Python)

```
sql = "SELECT user_id, user_name FROM users"
try:
    cnx = connect(user=username, password=passwd, host=hostname, database=db_name)
    cnx.autocommit = True
    cursor = cnx.cursor()

    cursor.execute(sql)
    for (id, name) in cursor:
        print("{} ma identyfikator {}".format(name, id))

    cursor.close()
    cnx.close()
except:
      # Obsługa błędu
```

```
Wojtek ma identyfikator 1
Wojtek2 ma identyfikator 2
Paweł ma identyfikator 3
Janusz ma identyfikator 4
```

Wczytywanie elementów (Python)

Możemy sobie zażyczyć słownika zamiast krotki w wyniku. Taki słownik będzie zawierał wszystkie pola z zapytania w formacie klucz – wartość.

W tym celu przy pobieraniu kursora należy wywołać metodę `cursor` z parametrem `cursor_factory=RealDictCursor`:

```
cnx.cursor(cursor_factory=RealDictCursor)
```

(Klasa `RealDictCursor` znajduje się w module `psycopg2.extras`)

```
from psycopg2.extras import RealDictCursor
sql = ("SELECT user_id, user_name"
       "FROM users")
cnx = connect(user=u, password=p,
              host=h, database=db)
cnx.autocommit = True
crs = cnx.cursor(
        cursor_factory=RealDictCursor)

crs.execute(sql)
for row in crs:
    print(row)
```

Wynik:

```
{"user_id": 1, "name": "Wojtek"}
{"user_id": 2, "name": "Wojtek2"}
{"user_id": 3, "name": "Paweł"}
{"user_id": 4, "name": "Janusz"}
```

Zmiana wartości danych

```
UPDATE users
SET user_name='Grzesiek'
WHERE user_id=2;
SELECT * FROM users;
```

```
---------+-----------
 user_id | user_name
---------+-----------
       1 | Wojtek
       2 | Grzesiek
         ...
---------+-----------
(4 rows)
```

Python

```
sql = """UPDATE users
         SET user_name='Grzesiek'
         WHERE user_id=2"""
try:
    cnx = connect(user=u, password=p,
                  host=h, database=db)
    cnx.autocommit = True
    cursor = cnx.cursor()

    cursor.execute(sql)
    print("Wpis został poprawiony")
except:
    print("Błąd!")
```

Usuwanie danych z tabeli

```
DELETE FROM users
WHERE user_name='Grzesiek';
SELECT * FROM users;
```

```
---------+-----------
 user_id | user_name
---------+-----------
       1 | Wojtek
       3 | Paweł
       4 | Janusz
---------+-----------
(3 rows)
```

```
sql = """DELETE FROM users
         WHERE user_name='Grzesiek'"""
try:
    cnx = connect(user=u, password=p,
                  host=h, database=db)
    cnx.autocommit = True
    cursor = cnx.cursor()

    cursor.execute(sql)
    print("Wpis został poprawiony")
except:
    print("Błąd!")
```