Języki skryptowe
Python

Uniwersytet Technologiczno-Przyrodniczy im. J.J. Śniadeckich w Bydgoszczy			
Instrukcja do ćwiczeń laboratoryjnych			
Przedmiot	**Nr**	**Temat**	**Studia**
Języki skryptowe	5	Bazy danych w Python	SP-1

Ćwiczenie 1. Tworzenie bazy danych w Pythonie z wykorzystaniem interpretera sqlite

1. **Połączenie z bazą danych.** W wybranym edytorze tworzymy plik sqltest.py i umieszczamy w nim poniższy kod.

```
#!/usr/bin/python3
import sqlite3

# utworzenie połączenia z bazą przechowywaną na dysku lub w pamięci (':memory:')
con = sqlite3.connect('test1.db')

# dostęp do kolumn przez indeksy i przez nazwy
con.row_factory = sqlite3.Row

# utworzenie obiektu kursora, istotne aby móc wykonywać operacje na bazie
cur = con.cursor()
```

2. **Model bazy danych.** Prawidłowe wykonywanie relacji na bazie danych, typu: Create, Read, Update, Delete wymaga utworzenia tabel i relacji między nimi, wg ustalonego schematu. Do podanego powyżej kodu należy dodać:

```
# tworzenie tabel
cur.execute("DROP TABLE IF EXISTS klasa;")

cur.execute("""
   CREATE TABLE IF NOT EXISTS klasa (
      id INTEGER PRIMARY KEY ASC,
      nazwa varchar(250) NOT NULL,
      profil varchar(250) DEFAULT ''
   )""")

cur.executescript("""
   DROP TABLE IF EXISTS uczen;
   CREATE TABLE IF NOT EXISTS uczen (
```

```
    id INTEGER PRIMARY KEY ASC,
    imie varchar(250) NOT NULL,
    nazwisko varchar(250) NOT NULL,
    klasa_id INTEGER NOT NULL,
    FOREIGN KEY(klasa_id) REFERENCES klasa(id)
  )""")
```

Pojedyncze polecenia SQL-a wykonujemy za pomocą metody .execute() obiektu kursora. Wiele instrukcji wykonujemy z wykorzystaniem metody .executescript().

Powyższe polecenia SQLa tworzę dwie tabele. Pierwszą o nazwie **klasa**, gdzie zawarto nazwę i profil klasy. Drugą o nazwie **uczeń**, która zawiera pola imię, nazwisko, identyfikator klasy. Między tabelami zachodzi relacja jeden-do-wielu., co oznacza, że do jednej klasy może przystępować wielu uczniów.

Po wykonaniu wprowadzonego kody w katalogu ze skryptem powinien się pojawić plik test.db, czyli utworzona baza danych. Jej zawartość będziemy omawiać przy użyciu *interpretera sqlite3*.

3. Interpreter sqlite3.

```
sandra@Ami ~/Pulpit $ sqlite3 test1.db
SQLite version 3.8.2 2013-12-06 14:53:30

Enter ".help" for instructions
Enter SQL statements terminated with a ";"

sqlite> .schema

sqlite> .database

seq  name            file
---  --------------  ----------------------------------------------------------
0    main            /home/sandra/Pulpit/test1.db
1    tem

sqlite> .schema
```

Opracowanie: mgr inż. Sandra Śmigiel
ZTC WTIE

```
sqlite> CREATE TABLE klasa (
   ...> id INTEGER PRIMARY KEY ASC,
   ...> nazwa varchar(250) NOT NULL,
   ...> profil varchar(250) DEFAULT"
   ...> );

sqlite> CREATE TABLE uczeń(
   ...> id INTEGER PRIMARY KEY ASC,
   ...> imie varchar(250) NOT NULL,
   ...> klasa_id INTEGER NOT NULL,
   ...> FOREIGN KEY(klasa_id) REFERENCES klasa(id)
   ...> );

sqlite> .tables
klasa   uczeń

sqlite> .quite
```

4. Wstawianie danych. Do utworzonego już skryptu dodajemy poniższy kod:

```
# wstawiamy jeden rekord danych
cur.execute('INSERT INTO klasa VALUES(NULL, ?, ?);', ('1A', 'matematyczny'))
cur.execute('INSERT INTO klasa VALUES(NULL, ?, ?);', ('1B', 'humanistyczny'))
# wykonujemy zapytanie SQL, które pobierze id klasy "1A" z tabeli "klasa".
cur.execute('SELECT id FROM klasa WHERE nazwa = ?', ('1A',))
klasa_id = cur.fetchone()[0]

# rekord "uczniowie" zawiera pola z danymi poszczególnych uczniów
uczniowie = (
    (None, 'Tomasz', 'Nowak', klasa_id),
    (None, 'Jan', 'Kos', klasa_id),
    (None, 'Piotr', 'Kowalski', klasa_id)
)
```

Opracowanie: mgr inż. Sandra Śmigiel
ZTC WTIE

```
# wstawiamy wiele rekordów
cur.executemany('INSERT INTO uczen VALUES(?,?,?,?)', uczniowie)

# zatwierdzamy zmiany w bazie
con.commit()
```

Wstawianie pojedynczych rekordów następuje z wykorzystaniem odpowiednim poleceń SQLa uznawanych za argumenty omawianej wcześniej metody.execute().

Wiele rekordów dodajemy analogicznie posługując się funkcją: .executescript().

Wartości pól nie należy umieszczać bezpośrednio w zapytaniu SQL! W ich miejsce wstawiono tzw. placeholder.

Metoda .commit() zatwierdza wprowadzone dane, tzn. zapisuje je w bazie danych.

5. **Pobieranie danych. Wymaga zastosowania polecenia SELECT. Do utworzonego już kodu dopisujemy funkcję, która wyświetli listę uczniów oraz klas, do których należą:**

```
def czytajdane():

    # Funkcja wybiera dane wszystkich uczniów z bazy i wyświetla je.
    cur.execute(
        'SELECT uczen.id,imie,nazwisko,nazwa FROM uczen,klasa WHERE uczen.klasa_id=klasa.id')
    uczniowie = cur.fetchall()
    for uczen in uczniowie:
        print uczen['id'], uczen['imie'], uczen['nazwisko'], uczen['nazwa']
    print ""

czytajdane()
```

Funkcja czytajdane() wykonuje zapytanie SQL pobierające wszystkie dane z dwóch powiązanych tabel „uczen” i „klasa”. Wydobywamy dzięki temu id ucznia, imię i nazwisko, a także nazwę klasy na podstawie warunku w klauzuli WHERE.

Metoda .fetchall() zwraca wynik, czyli pasujące rekordy do zapytania, które zapisywane są do zmiennej uczniowie, gdzie będą przechowywane. Elementy w niej zawarte odczytujemy w pętli for, jako listę uczen.

Opracowanie: mgr inż. Sandra Śmigiel
ZTC WTIE

6. Modyfikacje i usuwanie danych. Do skryptu dodaj następujący kod:

```
# zmiana klasy ucznia o identyfikatorze 2
cur.execute('SELECT id FROM klasa WHERE nazwa = ?', ('1B',))
klasa_id = cur.fetchone()[0]

cur.execute('UPDATE uczen SET klasa_id=? WHERE id=?', (klasa_id, 2,))

# usunięcie ucznia o identyfikatorze 3
cur.execute('DELETE FROM uczen WHERE id=?', (3,))

czytajdane()
con.close()
```

Modyfikowanie zawartych w klasie danych ucznia wymaga znajomości jego identyfikatora. Proces ten odbywa się z wykorzystaniem metody .execute() i odpowiednim poleceniem SELECT SQLa. Później konstruujemy zapytanie UPDATE aby nadać jej nowe wartości. Usuniecie danych ucznia następuje z wykorzystaniem polecenia SQL DELETE. Proces zamyka metoda .close().

Ćwiczenie 2. Tworzenie bazy danych w Pythonie bez interpretera.

1. Połączenie z bazą danych. Dwie możliwości zapisu danych – pamięć/plik w danym katalogu.

```
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test2.db')
print "Opened database successfully";
```

Prawidłowe wykonanie zadania zwróci w efekcie końcowym „Opened database successfully”;

2. Utworzenie tabeli o nazwie Company. Nadanie jej podstawowych cech.

```
#!/usr/bin/python

import sqlite3
```

```
conn = sqlite3.connect('test2.db')
print "Opened database successfully";

conn.execute('''CREATE TABLE IF NOT EXISTS COMPANY
      (ID INT PRIMARY KEY     ASC,
      NAME           TEXT    NOT NULL,
      AGE            INT     NOT NULL,
      ADDRESS        CHAR(50),
      SALARY         REAL);''')
print "Table created successfully";
conn.close()
```

Prawidłowe wykonanie zadania zwróci w efekcie końcowym „Opened database successfully” oraz „Table created successfully”;

3. Funkcja INSERT. Wypełnienie rekordów w bazie danych.

```
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test2.db')

print "Opened database successfully";

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (1, 'Paul', 32, 'California', 20000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (2, 'Allen', 25, 'Texas', 15000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (3, 'Teddy', 23, 'Norway', 20000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (4, 'Mark', 25, 'Rich-Mond ', 65000.00 )");
```

Opracowanie: mgr inż. Sandra Śmigiel
ZTC WTIE

```
conn.commit()

print "Records created successfully";

conn.close()
```

Prawidłowe wykonanie zadania zwróci w efekcie końcowym dwie linie:
Opened „Opened database successfully”
Records „Table created successfully”

4. Funkcja SELECT. Wybór z bazy pracowników po ID.

```
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test2.db')

print "Opened database successfully";

cursor = conn.execute("SELECT id, name, address, salary  from COMPANY")

for row in cursor:
   print "ID = ", row[0]
   print "NAME = ", row[1]
   print "ADDRESS = ", row[2]
   print "SALARY = ", row[3], "\n"

print "Operation done successfully";

conn.close()
```

5. Funkcja UPDATE uaktualnia dane zawarte w tabeli, wymaga znajomości ID.

```
#!/usr/bin/python
```

```
import sqlite3

conn = sqlite3.connect('test2.db')

print "Opened database successfully";

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

conn.commit

print "Total number of rows updated :", conn.total_changes

cursor = conn.execute("SELECT id, name, address, salary  from COMPANY")

for row in cursor:
   print "ID = ", row[0]
   print "NAME = ", row[1]
   print "ADDRESS = ", row[2]
   print "SALARY = ", row[3], "\n"

print "Operation done successfully";

conn.close()
```

6. Funkcja DELETE usuwa rekordy z tabeli, wymaga znajomości ID.

```
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect('test2.db')

print "Opened database successfully";
```

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

conn.commit

print "Total number of rows deleted :", conn.total_changes

cursor = conn.execute("SELECT id, name, address, salary  from COMPANY")

for row in cursor:
   print "ID = ", row[0]
   print "NAME = ", row[1]
   print "ADDRESS = ", row[2]
   print "SALARY = ", row[3], "\n"

print "Operation done successfully";

conn.close()
```

Zadanie problemowe. Utwórz bazę osób zawierającą: id, imię, datę urodzenia, zarobki. Napisz program z wykorzystaniem pętli, w którym zawrzesz następujące polecenia:
1 – wyświetlenie osób posortowanych alfabetycznie po imieniu
2 – dodanie nowej osoby
3 – modyfikowanie wpisu (np. zmiana daty urodzenia? zarobków? - dowolnie)
4 – usunięcie osoby
5 – wyświetlenie osób, które mają wiek w konkretnym przedziale
6 – wyświetlenie najwyższych zarobków
7 – pomysł własny
8 – wyjście

Opracowanie: mgr inż. Sandra Śmigiel
ZTC WTIE