České vysoké učení technické v Praze FIT

Programování v Pythonu

Jiří Znamenáček

Příprava studijního programu Informatika je podporována projektem financovaným z Evropského sociálního fondu a rozpočtu hlavního města Prahy.

Praha & EU: Investujeme do vaší budoucnosti

Python – AXML – SQLite v Python'u

SQLite v Python'u

Python od verze 2.5 obsahuje nativně SQLite. Přístup k ní zajišťuje modul `sqlite3`, pomocí kterého se k vybrané databázi připojíme, vykonáme svoji práci a na konci po sobě zase pěkně uklidíme. Veškeré operace s databází přitom zajišťuje speciální objekt `cursor`, který bychom po skončení práce měli také uzavřít. Typická seance s databází v Python'u vypadá tedy takto:

```
# encoding: utf-8

import sqlite3

# připojení k SQLite-databázi
# ~ pokud soubor neexistuje, bude v daném umístění vytvořen
connection = sqlite3.connect("structures.db")
# získání kurzoru
cursor = connection.cursor()

# práce s databází
...

# uzavření kurzoru
cursor.close()
# uzavření připojení k databázi
connection.close()
```

Kurzorů do databáze můžeme otevřít více najednou a pracovat s každým zvlášť nezávisle na ostatních.

Důležité: SQLite, narozdíl např. od MySQL, přijaté příkazy na změnu dat v databázi (tj. např. `INSERT`, `ALTER`, `UPDATE`…) automaticky neprovádí ihned! Chceme-li je tedy do databáze skutečně zaznamenat, musíme si to vyžádat vyvoláním metody `connection.commit()` na otevřené konexi.

Příklad – vyhledávání

V následujícím příkladu si ukážeme jednoduchý dotaz nad databází structures.db – vyhledání všech záznamů, jež ve jménu obsahují podřetězec „benzene“, aneb v SQL:

```
SELECT id, name FROM structures
    WHERE name LIKE '%benzene%';
```

Odpovídající program v Python'u je následující:

Program *structures.sqlite/structures.py*: **Výstup:**

```
# encoding: utf-8

import sqlite3

# připojení k SQLite-databázi
# ~ pokud soubor neexistuje, bu
connection = sqlite3.connect("s
# získání kurzoru („pracovního
cursor = connection.cursor()

# dotaz do databáze
cursor.execute("""
    SELECT id, name FROM struct
        WHERE name LIKE '%benze
    """)
# kurzor se chová jako iterátor
for id, name in cursor:
  print "%-30s %s" % ( name, "h

# uzavření kurzoru
cursor.close()
# uzavření připojení k databázi
connection.close()
```

```
benzene-1,2,3,5-tetrol
1,2,4-trichlorobenzene
benzene
3-phenylbenzene-1,2-diol
benzene-1,2-diol
2-chlorobenzene-1,4-diol
3-methylbenzene-1,2-diol
benzene-1,3,5-triol
benzene-1,4-diol
```

Odmyslíme-li si povinnou „omáčku“ připojovacích a uzavíracích příkazů (viz předchozí slajd), redukuje se práce s databází v Python'u na volání metody `cursor.execute('SQL-dotaz')` právě používaného kurzoru. Jejím parametrem je (jeden) konkrétní SQL-dotaz, kterým je naprosto standardní SQL příslušné databáze (zde tedy SQLite).

→ Zde dotaz `SELECT` nemění data v databázi, proto nemusíme volat `connection.commit()`.

Vytvoření databáze

Uvedená databáze structures.db byla vytvořena z textového souboru s daty structures.txt pomocí následujícího skriptu:

```
# encoding: utf-8

import sqlite3

# připojení k SQLite-databázi
# ~ pokud soubor neexistuje, bude v daném umístění vytvořen
connection = sqlite3.connect("structures.db")

# získání kurzoru
cursor = connection.cursor()

# odstranění případné již existující tabulky daného jména
cursor.execute("""
    DROP TABLE IF EXISTS structures;
    """)
# vytvoření tabulky
cursor.execute("""
    CREATE TABLE structures (
        id INTEGER PRIMARY KEY,
        name TEXT,
        inchikey TEXT,
        smiles TEXT);
    """)
# nucené vyvolání čekajících operací
# (to abychom nezačali s databází pracovat a ona ještě neexistoval
connection.commit()

# naplnění databáze/tabulky údaji ze souboru
f = file("structures.txt")
for line in f:
  id, inchikey, smiles, name = line.strip().split("\t")
  cursor.execute(
    "INSERT INTO structures (id,name,inchikey,smiles) VALUES (?,?,
    )
# „commit“ není třeba provádět po každém „execute“
# (prostě se INSERTů zavolá a vykoná více najednou)
connection.commit()

# uzavření kurzoru
cursor.close()
# uzavření připojení k databázi
connection.close()
```

Převedeno přímo do příslušného SQL jde tedy o následující sekvenci příkazů (zbývajících 197 INSERTů vynecháno :-):

```
DROP TABLE IF EXISTS structures;

CREATE TABLE structures (
    id INTEGER PRIMARY KEY,
    name TEXT,
    inchikey TEXT,
    smiles TEXT
    );

INSERT INTO structures (id,name,inchikey,smiles) VALUES (7,"MUIPLR
INSERT INTO structures (id,name,inchikey,smiles) VALUES (11,"WSLDO
INSERT INTO structures (id,name,inchikey,smiles) VALUES (12,"RDJUH
...
```

Za pozornost stojí zvláště použití znaku `?` pro vložení parametrů podle jejich pozice – za první otazníček se dosadí *id*, za druhý *name* atd.

Parametry

Parametry SQL-dotazu můžeme předávat pozičně jako v předchozím příkladu...

```
cursor.execute(
  "INSERT INTO structures (id,name,inchikey,smiles) VALUES (?,?,?,
)
```

...ale též pojmenovaně (což se hodí zvláště pro složitější příkazy):

```
cursor.execute(
  "INSERT INTO structures (id,name,inchikey,smiles) VALUES (:id,:n
  {"id": id, "name": name, "inchikey": inchikey, "smiles": smiles}
)
```

V předchozím příkladu jsme uvnitř smyčky neustále kolem dokola volali příkaz `INSERT INTO` pokaždé s jinými parametry. Pokud bychom parametry měli dopředu k dispozici ve formě datové struktury (nazvěme ji `data`), po které se dá iterovat, mohli bychom místo smyčky použít jednodušší příkaz:

```
cursor.executemany(
  "INSERT INTO structures (id,name,inchikey,smiles) VALUES (?,?,?,
  data
)
```

→ Užití viz modifikovaný příklad.

Více SQL-příkazů

Metoda `Cursor.execute()` přebírá **právě jeden** SQL-dotaz:

```
cursor.execute(
  "...SQL-1..."
)
```

Pro vykonání více SQL-příkazů najednou slouží metoda `Cursor.executescript()`:

```
cursor.executescript("""
  ...SQL-1...
  ...SQL-2...
  ...
""")
```

Vytvoření ukázkové databáze struktur tedy může vypadat také takto:

```
cursor.executescript("""
  DROP TABLE IF EXISTS structures;
  CREATE TABLE structures (
    id INTEGER PRIMARY KEY,
    name TEXT,
    inchikey TEXT,
    smiles TEXT);
""")
```

Získávání výsledků

Už jsme viděli, že výsledek SQL-dotazu do databáze se vrátí jako iterátor (tedy struktura, po které se dá iterovat):

```
# dotaz do databáze
cursor.execute("""
    SELECT id, name FROM structures
        WHERE name LIKE '%benzene%';
  """)
# kurzor se chová jako iterátor přes množinu výsledků dotazu
for name in cursor:
  print name
```

Někdy se nám však mohou hodit další varianty získání výsledů:

- `cursor.fetchone()` – vrátí následující řádek z návratové množiny dotazu (nebo `None`, pokud už tam další není)
- `cursor.fetchmany([size])` – vrací pouze zadaný (či menší, nejsou-li již k dispozici) počet řádek

> Uvedenou hodnotu je sice možno předat přímo jako parametr, ale z důvodu optimalizace je lepší ji nastavit dopředu jako atribut daného kurzoru (tj. pomocí `cursor.arraysize = size`). Ve druhém případě se pak volá přímo `cursor.fetchmany()` bez parametrů.

- `cursor.fetchall()` – vrátí všechny zbývající řádky (od aktuální pozice) z návratové množiny dotazu jako seznam (nejsou-li žádné řádky k dispozici, vrácený seznam bude prázdný)

Mapování typů

Python (< 3.0)	Python (>= 3.0)	SQLite
None	None	NULL
int	int	INTEGER
long	–	INTEGER
float	float	REAL
str (UTF-8)	str	TEXT
unicode	–	TEXT
buffer	bytes	BLOB

SQLite má typy nejen velmi jednoduché, ale navíc i dynamické (typ je vázán na konkrétní údaj, nikoli na místo uložení v databázi, tj. „sloupeček“). To je ale mezi databázemi výjimka (stejně jako u programovacích jazyků se dynamické typy špatně optimalizují a blbě ladí) a rozhodně si na to nezvykejte.

Jednoduchosti bohužel padl za oběť třeba i typ `Boolean`, který je při standardní konverzi ukládán jako 0/1. (Naše ukázková databáze world převedená z MySQL používá jako pozůstatek konverze F/T.)

Na druhou stranu – pokud o to budete hodně stát, můžete SQLite v Python'u naučit používat (tedy konvertovat na jednoduché typy) i další pythonovské typy než pouze výše uvedené základní.

„Zkratky“

Pokud potřebujete rychle získat data z databáze či provést nějakou jednoduchou úpravu, nabízí pythonovské rozhraní k SQLite-databázi (DB

API 2) několik užitečných zkratek – všechny tři varianty metody *execute* lze totiž volat přímo na otevřeném připojení do databáze:

```
import sqlite3
connection = sqlite3.connect("structures.db")

connection.execute("...sql...")
connection.executemany("...sql...", params)
connection.executescript("...sql...")
```

Není-li tudíž potřeba použití kurzorů z jiného důvodu, můžete se jim úplně vyhnout – uvedené „zkratky“ si pro vykonání zadaného SQL-příkazu(ů) vyrobí dočasný kurzor samy.