POGLAVLJE 6

Programiranje baza podataka

Zaista ste svom sinu dali ime `Robert'); DROP TABLE Students;`*?*
– Rendal Manro, veb strip XKCD na vebu, oktobar 2007.

U ovom poglavlju...

- Uvod
- DB-API Pythona
- Objektno-relacioni maperi
- Nerelacione baze podataka
- Reference

U ovom poglavlju objašnjavamo kako komunicirati s bazama podataka pomoću Pythona. Datoteke ili jednostavna trajna skladišta mogu da zadovolje potrebe manjih aplikacija, ali većem serveru ili aplikacijama koje koriste velike količine podataka možda će biti potreban ceo sistem baze podataka. Predstavićemo i relacione i nerelacione baze podataka kao i objektno-relacione mapere (engl. *Object-Relational Mapper, ORM*).

6.1 Uvod

U uvodnom odeljku razmatramo potrebu za bazama podataka, predstavljamo strukturirani jezik upita (engl. *Structured Query Language, SQL*), i upoznajemo čitaoce s Pythonovim interfejsom za programiranje aplikacija za rad s bazama podataka (API).

6.1.1 Trajno skladište

Svaka aplikacija ima potrebu za trajnim skladištem. U načelu, postoje tri osnovna mehanizma skladištenja: datoteka, sistem baze podataka ili neki hibrid, poput API-ja povrh nekog od tih postojećih sistema, ORM, upravljač datotekama (engl. *file manager*), proračunska tabela, konfiguraciona datoteka itd.

U poglavlju „Datoteke“ knjige *Core Python Language Fundamentals* ili *Core Python Programming*, govorili smo o trajnom skladištenju pomoću prostog pristupa datoteci kao i Pythonovog upravljača i upravljača bazama podataka (engl. *database manager, DBM*), koji predstavlja stari Unixov mehanizam trajnog skladištenja povrh datoteka (`*dbm`, `dbhash/bsddb`), `shelve` (kombincija modula `pickle` i upravljača bazom podataka), i pomoću njihovog interfejsa objekata sličnog rečniku.

U ovom poglavlju fokusiraćemo se na korišćenje baza podataka u situacijama kada datoteke ili pravljenje sopstvenih sistema skladištenja podataka nisu dovoljni za veće projekte. U takvim slučajevima, moraćete da donesete mnogo odluka. Cilj ovog poglavlja je, zato, da vas upozna sa osnovama i da vam pokaže koliko opcija imate na raspolaganju (i kako raditi s njima u okviru Pythona) kako biste mogli da donesete pravu odluku. Na početku, predstavljamo SQL i relacione baze podataka, jer je to i dalje prevlađujući oblik trajnog skladišta.

6.1.2 Osnovne operacije nad bazama podataka i SQL

Pre nego što se bacimo na baze podataka i pokažemo kako se koriste s Pythonom, ukratko ćemo se osvrnuti na neke elementarne koncepte baza podataka i SQL.

Osnovno skladište

Baze podataka obično imaju neko osnovno trajno skladište koje koristi sistem datoteka, to jest, normalne datoteke operativnog sistema, specijalne datoteke operativnog sistema, čak i sirove particije diska.

Korisnički interfejs

Većina sistema baza podataka ima alatku komandne linije pomoću koje se izdaju SQL naredbe ili upiti. Postoje i GUI alatke koje koriste komandnu liniju za klijenta ili klijentske biblioteke baza podataka, što korisnicima pruža mnogo pogodniji interfejs.

Baze podataka

Sistem upravljanja relacionom bazom (engl. *relational database management system, RDBMS*) obično može da upravlja većim brojem baza podataka – na primer, prodajom, marketingom, korisničkom podrškom itd. – na istom serveru (ukoliko je RDBMS zasnovan na serveru, što jednostavniji sistemi obično nisu). U primerima iz ovog poglavlja, MySQL će biti ilustracija za RDBMS zasnovan na serveru, jer se proces servera neprestano izvršava, čekajući na naredbe; ni SQLite ni Gadfly nemaju servere koji se stalno izvršavaju.

Komponente

Tabela je apstrakcija skladišta za bazu podataka. Svaki *red* podataka ima polja koja odgovaraju *kolonama* baze podataka. Skup definicija kolona tabele i tipova podataka po tabeli zajedno definišu *šemu* baze podataka.

Baze podataka *pravimo* i *brišemo* (engl. *drop*). Isto važi za tabele. Dodavanje novih redova bazi zove se *umetanje* (engl. *inserting*), promena postojećih redova u tabeli je *ažuriranje* (engl. *updating*), dok je uklanjanje postojećih redova u tabeli *brisanje* (engl. *deleting*). Ove aktivnosti se obično zovu *naredbe* ili *operacije* nad bazama podataka. Traženje redova iz baze podataka pomoću opcionih kriterijuma zove se *izvršavanje upita* (engl. *querying).*

Kada izvršite upit nad bazom padataka, možete *dobaviti* sve rezultate (redove) odjednom, ili polako redom pristupati svakom redu u rezultatu. Pojedine baze podataka koriste koncept *kursora* za davanje SQL naredbi, upita i dobavljanje rezultata, bilo odjednom ili red po red.

SQL

Naredbe i upiti se bazi podataka zadaju putem SQL-a. Ne koriste sve baze SQL, ali većina relacionih baza podataka ga podržava. Sada ćemo navesti nekoliko primera SQL naredbi. Većina baza podataka je podešena tako da nema razlike između velikih i malih slova. Usvojeno je da se za ključne reči baze podataka koriste velika slova. Većina programa komandne linije traži tačku i zarez (`;`) na kraju SQL naredbe.

Pravljenje baze podataka

```
CREATE DATABASE test;
GRANT ALL ON test.* to user(s);
```

U prvom redu pravimo bazu podataka pod imenom „test“, i, pod pretpostavkom da ste administrator baze podataka, drugi red možete iskoristiti da date pravo pristupa određenim (ili svim) korisnicima, tako da mogu da obavljaju naredne operacije nad bazom podataka.

Korišćenje baze podataka

```
USE test;
```

Ako ste se prijavili u sistem baza podataka ne odabravši koju ćete bazu podataka da koristite, ovaj jednostavan iskaz omogućava da navedete nad kojom bazom ćete obavljati operacije.

Uklanjanje baze podataka

```
DROP DATABASE test;
```

Ovom jednostavnim naredbom uklanjaju se sve tabele i podaci iz baze podataka, i ona se briše iz sistema.

Pravljenje tabele

```
CREATE TABLE users (login VARCHAR(8), userid INT, projid INT);
```

Ova naredba pravi novu tabelu, s kolonom `login` tipa znakovnog niza i parom celobrojnih polja, `userid` i `projid`.

Uklanjanje tabele

```
DROP TABLE users;
```

Ovom jednostavnom naredbom uklanja se tabela iz baze podataka zajedno sa svim njenim podacima.

Umetanje reda

```
INSERT INTO users VALUES('leanna', 2111, 1);
```

U tabelu možete umetnuti novi red pomoću naredbe `INSERT`. Navodite tabelu i vrednosti za svako polje. U našem primeru, znakovni niz `'leanna'` upisuje se u polje `login`, a vrednosti 2111 i 1 u polja `userid` odnosno `projid`.

Ažuriranje reda

```
UPDATE users SET projid=4 WHERE projid=2;
UPDATE users SET projid=1 WHERE userid=311;
```

Da biste izmenili postojeće redove tabele, upotrebićete naredbu `UPDATE`. Naredba `SET` se koristi da izmenite kolone, pri čemu navodite kriterijum za određivanje redova koji treba da se izmene. U prvom primeru, svi korisnici čiji ID projekta (ili `projid`) ima vrednost 2 biće premešteni u projekat #4. U drugom primeru, jednog korisnika (čiji korisnički ID ima vrednost 311) premestićemo u projekat 1.

Brisanje reda

```
DELETE FROM users WHERE projid=%d;
DELETE FROM users;
```

Ako želite da izbrišete red tabele, upotrebićete naredbu `DELETE FROM`, navodeći tabelu iz koje želite da izbrišete redove i bilo koji opcioni kriterijum. Bez toga, izbrisaće se svi redovi, kao u drugom primeru.

Sada, pošto ste savladali osnovne koncepte baze podataka, ostatak poglavlja i preostali primeri biće vam mnogo lakši. Ako vam je potrebna dodatna pomoć, postoji mnogo knjiga o bazama podataka iz kojih ćete saznati ono što želite.

6.1.3 Baze podataka i Python

U ovom odeljku predstavićemo Pythonov API za baze podataka i objasniti kako se pristupa relacionim bazama podataka iz Pythona – bilo direktno preko interfejsa baze podataka, ili pomoću objektno-relacionog mapera – i kako možete da obavite isti zadatak bez potrebe da zadajete eksplicitne SQL naredbe.

Pitanja kao što su principi baza podataka, paralelni rad, šema, nedeljivost, integritet, oporavljanje, složeni levi spojni upiti, okidači, optimizacija upita, transkacije, uskladištene procedure itd., prevazilaze opseg ovog teksta, i u ovom poglavlju nećemo im se posvetiti, izuzev kad ih budemo direktno primenjivali u Pythonovoj aplikaciji. Umesto toga, objasnićemo kako da čuvate i dobavljate podatke u okviru RDBMS sistema, dok se igramo u Pythonovom okruženju. Posle toga moći ćete da odlučite šta je najbolje za vaš tekući projekat ili aplikaciju, i da analizirate primere koda koji vas lako mogu nadahnuti. Cilj je da što pre shvatite kako stvari funkcionišu, ako je potrebno da svoju aplikaciju napisanu na Pythonu integrišete s nekom vrstom sistema baze podataka.

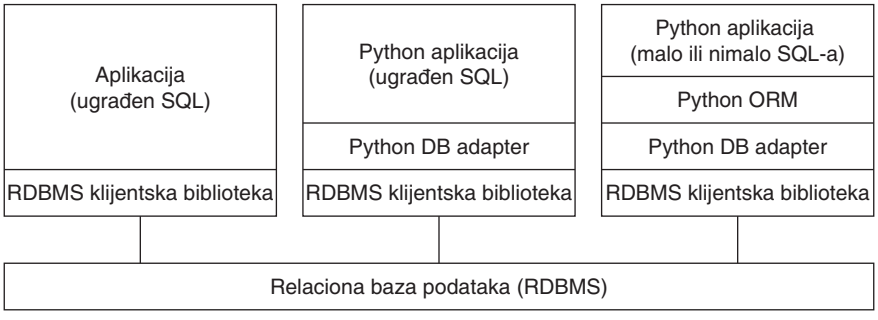
Uz to, prekršićemo sopstveno pravilo da predstavljamo samo elemente standardne biblioteke Pythona. Iako nam je cilj bio da se fokusiramo samo na to, jasno je da je rad s bazama podataka zapravo jedna od ključnih komponenti svakodnevnog razvoja aplikacija u Pythonovom svetu.

U vašoj karijeri softverskog inženjera, postoji trenutak kada više nećete moći da napredujete ako niste naučili neke stvari o bazama podataka: kako se koriste (iz komandne linije i/ili sa GUI interfejsima), kako se dobavljaju podaci iz nje pomoću SQLa, možda kako se dodaju ili ažuriraju podaci u bazi podataka itd. Ako Python jeste vaša alatka za programiranje, mnogo teškog rada je već obavljeno umesto vas, kada pridodate pristup bazama podataka svom Python univerzumu. Prvo objašnjavamo šta je Pythonov standard za programiranje aplikacija za rad s bazama podataka (*DB-API)*, potom dajemo primere interfejsa baza podataka koji odgovaraju tom standardu.

Pokazaćemo vam neke primere u kojima se koriste popularni RDBMS sistemi otvorenog koda. Međutim, nećemo se baviti poređenjem proizvoda otvorenog koda i komercijalnih proizvoda. Prilagođavanje tim drugim RDBMS sistemima trebalo bi da bude prilično jednostavno. Posebno ćemo se osvrnuti na Gadfly autora Arona Vatersa, jednostavan RDBMS napisan u potpunosti na Pythonu.

Bazi podataka se iz Pythona pristupa preko *adaptera*. Adapter je Pythonov modul pomoću kog možete uspostaviti interfejs za klijentsku biblioteku relacionih baza podataka (obično na C-u). Preporučljivo je da svi Pythonovi adapteri budu usklađeni sa okruženjem za programiranje aplikacija Pythonove posebne interesne grupe za baze podataka (engl. *database special interest group, DB-SIG*). To je prva glavna tema u ovom poglavlju.

Na slici 6-1 predstavljeni su slojevi pisanja Pythonove aplikacije za bazu podataka, sa i bez mapera ORM. Slika pokazuje da je DB-API vaš interfejs za C biblioteke klijenta baze podataka.



Slika 6-1 Višeslojna komunikacija između aplikacije i baze podataka. Prvi okvir je, načelno, C/C++ program, dok adapteri usklađeni sa standardom DB-API omogućavaju da programirate aplikacije na Pythonu. ORM može da pojednostavi aplikaciju starajući se o detaljima vezanim isključivo za bazu podataka.

6.2 Pythonov DB-API

Gde naći interfejse potrebne za komuniciranje s bazom podataka? To je lako. Dovoljno je da posetite odeljak o bazama podataka na glavnoj Pythonovoj veb lokaciji. Tu ćete naći veze ka potpunoj i aktuelnoj specifikaciji DB-API (verzija 2.0), postojećim modulima baze podataka, dokumentaciji, posebnoj interesnoj grupi itd. Još od nastanka, DB-API je prebačena u PEP 249. Ovaj PEP je potisnuo staru specifikaciju DB-API 1.0 (PEP 248). Šta je DB-API?

DB-API je specifikacija koja navodi skup potrebnih objekata i mehanizama pristupa bazama podataka kako bi se uspostavio usaglašen pristup za sve adaptere i sisteme baza podataka. Poput većine projekata nastalih angažovanjem zajednice, DB-API je rezultat jake potrebe.

U „stara vremena“, bilo je mnogo baza podataka i ljudi koji su implementirali sopstvene adaptere za baze podataka. Bilo je to poput tople vode koja se neprestano izmišljala. Te baze podataka i adaptere implementirali su različiti ljudi u raznim trenucima, bez ikakve konzistentnosti u funkcionalnosti. Nažalost, to je značilo da je kôd aplikacije koji koristi takve interfejse takođe morao da bude prilagođen odabranom modulu baze podataka, pa je svaka izmena interfejsa zahtevala ažuriranje koda aplikacije.

Formirana je posebna interesna grupa za povezanost Pythona s bazama podataka, i rodio se API: DB-API verzija 1.0. Ovaj API obezbeđuje konzistentan interfejs za razne relacione baze podataka, a prilagođavanje koda različitim bazama podataka mnogo je jednostavnije, i obično traži da izmenite tek po koji red koda. Videćete primer toga kasnije u ovom poglavlju.

6.2.1 Atributi modula

Specifikacija za DB-API nalaže da se moraju obezbediti dole navedeni atributi i elementi. Modul usklađen sa specifikacijom DB-API mora da definiše globalne atribute prikazane u tabeli 6-1.

Tabela 6-1 Atributi DB-API modula

Atribut	Opis
`apilevel`	Verzija specifikacije za DB-API s kojom je adapter usaglašen
`threadsafety`	Nivo sigurnosti niti ovog modula
`paramstyle`	Stil parametara SQL naredbe ovog modula
`connect()`	Funkcija `connect()`
(Razni izuzeci)	*(Videti tabelu 6-4)*

Atributi podataka

apilevel

Ovaj znakovni niz (a ne decimalni broj) ukazuje na najvišu verziju specifikacije za DB-API s kojom je modul usaglašen – na primer, 1.0, 2.0 itd. Ako se izostavi, treba pretpostaviti da je 1.0 podrazumevana vrednost.

threadsafety

Ceo broj koji može imati neku od narednih vrednosti:

- 0: Nije sigurno za niti, tako da niti ne bi trebalo da dele ovaj modul
- 1: Minimalno sigurno za niti: niti mogu da dele modul, ali ne konekcije
- 2: Umereno sigurno za niti: niti mogu da dele modul i konekcije, ali ne i kursore
- 3: Potpuno sigurno za niti: niti mogu da dele modul, konekcije i kursore

Ako se deli resurs, osnovni mehanizam sinhronizacije – poput brave ili semafora – neophodan je zbog atomskog zaključavanja. Datoteke sa diska i globalne promenljive nisu pouzdane i mogu da ometaju standardne operacije s muteksima. Ako želite više informacija o tome kako se koriste brave, pogledajte modul `threading` ili se vratite na poglavlje 4, „Višenitno programiranje“.

paramstyle

DB-API podržava razne načine da se ukaže koji parametri treba da se integrišu u SQL naredbu koja se na kraju šalje serveru na izvršenje. Ovaj argument je samo znakovni niz koji određuje oblik zamene znakovnog niza koji ćete upotrebiti kada budete pravili redove za svoj upit ili naredbu (videti tabelu 6-2).

Tabela 6-2 `paramstyle` stilovi parametra

Stil parametra	Opis	Primer
`numeric`	Numerilčki pozicioni stil	`WHERE name=:1`
`named`	Imenovan stil	`WHERE name=:name`
`pyformat`	Pythonova `printf()` konverzija formata	`WHERE name=%(name)s`
`qmark`	Stil znaka pitanja	`WHERE name=?`
`format`	ANSI C `printf()` konverzija formata	`WHERE name=%s`

Atributi funkcija

`connect()` Pristup funkcija bazi podataka omogućen je preko objekata tipa `Connection`. Usaglašen modul mora da implementira funkciju `connect()`, koja pravi i vraća objekat tipa `Connection`. Tabela 6.3 prikazuje argumente funkcije `connect()`.

Tabela 6-3 Atributi funkcije `connect()`

Parametar	Opis
`user`	Korisničko ime
`password`	Lozinka
`host`	Ime domaćina
`database`	Ime baze podataka
`dsn`	Ime izvora podataka

Informaciju za konekciju s bazom podataka možete da prosledite kao znakovni niz s više parametara (DSN) ili kao zasebne parametre prosleđene kao pozicione argumente (ukoliko znate tačan redosled). Evo primera s funkcijom `connect()` iz PEP 249:

```
connect(dsn='myhost:MYDB',user='guido',password='234$')
```

Da li ćete koristiti DSN ili zasebne parametre uglavnom zavisi od sistema s kojim se povezujete. Na primer, ako koristite API kao što je Open Database Connectivity (ODBC) ili Java DataBase Connectivity (JDBC), verovatno ćete primenjivati DSN, ali ukoliko radite direktno s bazom podataka, verovatnije je da ćete zadavati zasebne parametre prijavljivanja. Drugi razlog je to što većina adaptera baza podataka nije implementirala podršku za DSN. Navešćemo neke primere poziva funkcije `connect()` bez DSN-a. Uočite da nisu svi adapteri dosledno implementirali specifikaciju – na primer, `MySQLdb` koristi `db` umesto `database`.

- `MySQLdb.connect(host='dbserv', db='inv', user='smith')`
- `PgSQL.connect(database='sales')`
- `psycopg.connect(database='template1', user='pgsql')`
- `gadfly.dbapi20.connect('csrDB', '/usr/local/database')`
- `sqlite3.connect('marketing/test')`

Izuzeci

Izuzeci koje bi usaglašen modul morao da obuhvati prikazani su u tabeli 6-4.

Tabela 6-4 DB-API klase izuzetaka

Izuzetak	Opis
`Warning`	Korenska klasa izuzetka za upozorenje
`Error`	Korenska klasa izuzetka za grešku
`InterfaceError`	Greška interfejsa baze podataka (ne greška baze podataka)
`DatabaseError`	Greška baze podataka
`DataError`	Problemi sa obrađenim podacima
`OperationalError`	Greška prilikom izvršavanja operacije nad bazom podataka
`IntegrityError`	Greška relacionog integriteta baze podataka
`InternalError`	Greška unutar baze podataka
`ProgrammingError`	Neuspeh SQL naredbe
`NotSupportedError`	Došlo je do operacije koja nije podržana