6.8 与关系型数据库的交互¶

问题¶

你想在关系型数据库中查询、增加或删除记录。

解决方案¶

Python中表示多行数据的标准方式是一个由元组构成的序列。例如：

```
stocks = [
    ('GOOG', 100, 490.1),
    ('AAPL', 50, 545.75),
    ('FB', 150, 7.45),
    ('HPQ', 75, 33.2),
]
```

依据PEP249，通过这种形式提供数据， 可以很容易的使用Python标准数据库API和关系型数据库进行交互。 所有数据库上的操作都通过SQL查询语句来完成。每一行输入输出数据用一个元组来表示。

为了演示说明，你可以使用Python标准库中的 `sqlite3` 模块。 如果你使用的是一个不同的数据库(比如MySql、Postgresql或者ODBC)， 还得安装相应的第三方模块来提供支持。 不过相应的编程接口几乎都是一样的，除了一点点细微差别外。

第一步是连接到数据库。通常你要执行 `connect()` 函数， 给它提供一些数据库名、主机、用户名、密码和其他必要的一些参数。例如：

```
>>> import sqlite3
>>> db = sqlite3.connect('database.db')
>>>
```

为了处理数据，下一步你需要创建一个游标。 一旦你有了游标，那么你就可以执行SQL查询语句了。比如：

```
>>> c = db.cursor()
>>> c.execute('create table portfolio (symbol text, shares integer, price real)')
<sqlite3.Cursor object at 0x10067a730>
>>> db.commit()
>>>
```

为了向数据库表中插入多条记录，使用类似下面这样的语句：

```
>>> c.executemany('insert into portfolio values (?,?,?)', stocks)
<sqlite3.Cursor object at 0x10067a730>
>>> db.commit()
>>>
```

为了执行某个查询，使用像下面这样的语句：

```
>>> for row in db.execute('select * from portfolio'):
...     print(row)
...
('GOOG', 100, 490.1)
('AAPL', 50, 545.75)
('FB', 150, 7.45)
('HPQ', 75, 33.2)
>>>
```

如果你想接受用户输入作为参数来执行查询操作，必须确保你使用下面这样的占位符``?``来进行引用参数：

```
>>> min_price = 100
>>> for row in db.execute('select * from portfolio where price >= ?',
                          (min_price,)):
...     print(row)
...
```

```
('GOOG', 100, 490.1)
('AAPL', 50, 545.75)
>>>
```

讨论¶

在比较低的级别上和数据库交互是非常简单的。 你只需提供SQL语句并调用相应的模块就可以更新或提取数据了。 虽说如此，还是有一些比较棘手的细节问题需要你逐个列出去解决。

一个难点是数据库中的数据和Python类型直接的映射。 对于日期类型，通常可以使用 `datetime` 模块中的 `datetime` 实例， 或者可能是 `time` 模块中的系统时间戳。 对于数字类型，特别是使用到小数的金融数据，可以用 `decimal` 模块中的 `Decimal` 实例来表示。 不幸的是，对于不同的数据库而言具体映射规则是不一样的，你必须参考相应的文档。

另外一个更加复杂的问题就是SQL语句字符串的构造。 你千万不要使用Python字符串格式化操作符(如%)或者 `.format()` 方法来创建这样的字符串。 如果传递给这些格式化操作符的值来自于用户的输入，那么你的程序就很有可能遭受SQL注入攻击(参考 http://xkcd.com/327)。 查询语句中的通配符 `?` 指示后台数据库使用它自己的字符串替换机制，这样更加的安全。

不幸的是，不同的数据库后台对于通配符的使用是不一样的。大部分模块使用 `?` 或 `%s` ， 还有其他一些使用了不同的符号，比如:0或:1来指示参数。 同样的，你还是得去参考你使用的数据库模块相应的文档。 一个数据库模块的 `paramstyle` 属性包含了参数引用风格的信息。

对于简单的数据库数据的读写问题，使用数据库API通常非常简单。 如果你要处理更加复杂的问题，建议你使用更加高级的接口，比如一个对象关系映射ORM所提供的接口。 类似 `SQLAlchemy` 这样的库允许你使用Python类来表示一个数据库表， 并且能在隐藏底层SQL的情况下实现各种数据库的操作。