Database Connectivity: Python

Database Connecitivity Basics

Application-level database connectivity:

- Host language (Java, C/C++, Perl, PHP, etc)
- Target DBMS (Oracle, MS SQL server, IBM DB2, MySQL, etc)
- Client — Server environment
  - Client: application program
  - Server: DBMS

General structure:

- Load database driver/database support functionality
- form an SQL statement
- connect to the DBMS
- pass SQL statement to the DBMS
- recieve result
- close connection

Python Database Connectivity Standard

Python uses PEP 249: Python Database API Specification v.2.0 as the standard communication API for communicating with relational DBMS. This standard was adopted in 2001. The standard can be found here:

https://peps.python.org/pep-0249/

PEP 249 is implemented by a number of Python packages, e.g., MySQL Connector Python, PyMySQL, MySQLDB, and so on. We use MySQL Connector Python as our running example.

Before using DBMS connectivity, a package implementing PEP 249 needs to be installed for your version of Python. You can use

```
$ pip install mysql-connector-python
```

or

```
$ conda install mysql-connector-python
```

depending on your Python installation. You can also install this package through an interactive package manager.

Classes to import

With a PEP 249-compliant package installed, you need to import the `connector` class. You also, for the purpose of error management, may want to import the `Error` class.

```
import mysql.connector
from mysql.connector import Error
```

Connecting to MySQL

A SQL connection is created by calling a `connect()` method from the `mysql.connector` class and passing the settings for the connection. The result of the method is stored in a `connection` object.

The `connect()` method takes the following parameters:

Parameter	Meaning
`host`	name/address of the MySQL server host
`port`	port for access to the MySQL server
`database`	MySQL database on the server that will be accessed
`user`	user name for MySQL access
`password`	password for MySQL access

We always want to connect using a `try` block so that connection exceptions are gracefully handled.

Method `is_connected()` returns `True` if connection was successful.

We also need to close the connection at the end using the `close()` method for the `connection` objects.

The overall template for a Python script connecting to a DBMS looks as follows.

```
import mysql.connector
from mysql.connector import Error

try:
conn = mysql.connector.connect(host = 'mysql.labthreesixfive.com',
                                      port = '3306',
```

```
                                        database = 'alex',
                                        user = 'alex',
                                        password = '')

        if conn.is_connected():
             ### do your work with the DBMS here

except Error as e:
print("MySQL Connection error:",e)

finally:
if conn.is_connected():
                ## wrap up your work
conn.close()
```

Passing SQL commands

Python uses cursors for passing SQL commands. There are two types of SQL commands: those that do not return results (DML/DDL statements) and those that return tabular results (SQL SELECT statement).

A `cursor` object can be created by calling a `cursor()` method of the `connection` class.

```
cursor = conn.cursor()
```

The following methods can be used with cursors:

Method	Parameters	Meaning
`execute`	*statement* (string)	Execute a SQL statement
`fetchone()`		Return the next result from the result set produced by the executed SQL statement
`fetchall()`		Return all results a single result set