Bootcamp Training

Python

SummitWorks™
GLOBAL SOLUTION ARCHITECTS

Authorized & published by Summitworks Technologies Inc

Day -5: **Data Management**

- Best Practices for Data Management
  - Storing Data in Local Databases
  - Discussing and Understanding the DB API
  - Installing Drivers
  - Understanding and Using Common SQL Statements
  - Connecting to a SQL and no SQL Databases
  - Using Cursors to interact with Data from a Database
  - CRUD Operations
  - Invoking a Stored Procedure using python
  - Transaction management

Introduction

- The Python standard for database interfaces is the Python DB-API. Most Python database interfaces adhere to this standard.
- You can choose the right database for your application. Python Database API supports a wide range of database servers such as :
  - MySQL
  - PostgreSQL
  - Microsoft SQL Server 2000
  - MongoDB
  - Oracle
  - Sybase
  - SQLite

Introduction

- We have the list of Python database interfaces, please check https://wiki.python.org/moin/DatabaseInterfaces for list of database interface for python.
- You must download a separate DB API module for each database you need to access.

Python with MySQL

- **MySQLdb** is an interface for connecting to a MySQL database server from Python. It implements the Python Database API v2.0 and is built on top of the MySQL C API.
- You make sure you have MySQLdb installed on your machine.

Database Connection

```
import pymysql
# Open database connection
db = pymysql.connect("localhost","testuser","test123","TESTDB" )
# Prepare a cursor object using cursor() method
cursor = db.cursor()
# Execute SQL query using execute() method
cursor.execute("SELECT VERSION()")
# Fetch a single row using fetchone() method
data = cursor.fetchone()
print "Database version : %s " % data
# disconnect from server
db.close()
```

Output is :
Database version : 5.0.45

Creating Database Table

- Once a database connection is established, we are ready to create tables or records into the database tables using **execute** method of the created cursor.

```
import MySQLdb
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )
# prepare a cursor object using cursor() method
cursor = db.cursor()
# Drop table if it already exist using execute() method.
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")
# Create table as per requirement
sql = """CREATE TABLE EMPLOYEE (
         FIRST_NAME  CHAR(20) NOT NULL,
         LAST_NAME  CHAR(20),
         AGE INT,
         SEX CHAR(1),
         INCOME FLOAT )"""
cursor.execute(sql)
# disconnect from server
db.close()
```

READ Operation

- READ Operation on any database means to fetch some useful information from the database.
- Once our database connection is established, you are ready to make a query into this database. You can use either **fetchone()** method to fetch single record or **fetchall()** method to fetch multiple values from a database table.
- **fetchone():** It fetches the next row of a query result set. A result set is an object that is returned when a cursor object is used to query a table.
- **fetchall():** It fetches all the rows in a result set. If some rows have already been extracted from the result set, then it retrieves the remaining rows from the result set.
- **rowcount:** This is a read-only attribute and returns the number of rows that were affected by an execute() method.

Performing Transactions

Transactions are a mechanism that ensures data consistency. Transactions have the following four properties:

- **Atomicity:** Either a transaction completes or nothing happens at all.
- **Consistency:** A transaction must start in a consistent state and leave the system in a consistent state.
- **Isolation:** Intermediate results of a transaction are not visible outside the current transaction.
- **Durability:** Once a transaction was committed, the effects are persistent, even after a system failure.

The Python DB API 2.0 provides two methods to either *commit* or *rollback* a transaction.

Example

```
# Prepare SQL query to DELETE required records
sql = "DELETE FROM EMPLOYEE WHERE AGE > '%d'" % (20)
try:
   # Execute the SQL command
   cursor.execute(sql)
   # Commit your changes in the database
   db.commit()
except:
   # Rollback in case there is any error
   db.rollback()
```

COMMIT Operation

- Commit is the operation, which gives a green signal to database to finalize the changes, and after this operation, no change can be reverted back.
- Here is a simple example to call **commit** method.

db.commit()

ROLLBACK Operation

- If you are not satisfied with one or more of the changes and you want to revert back those changes completely, then use rollback() method.

- Here is a simple example to call rollback() method.

db.rollback()

Python with SQLite

- SQLite is a simple relational database system, which saves its data in regular data files or even in the internal memory of the computer, i.e. the RAM. It was developed for embedded applications, like Mozilla-Firefox (Bookmarks), Symbian OS or Android. SQLITE is "quite" fast, even though it uses a simple file.
- It can be used for large databases as well.
- If you want to use SQLite, you have to import the module sqlite3.
- To use a database, you have to create first a Connection object.
- The connection object will represent the database.
- SQLite interface is already available from python 2.5 onwards

Example

The connection object will represent the database. The argument of connection - in the above example "company.db" - functions both as the name of the file, where the data will be stored, and as the name of the database. If a file with this name exists, it will be opened. It has to be a SQLite database file of course! In the following example, we will open a database called company.

```
>>> import sqlite3
>>> connection = sqlite3.connect("company.db")
```

Example

```
CREATE TABLE employee (
staff_number INT NOT NULL AUTO_INCREMENT,
fname VARCHAR(20),
lname VARCHAR(30),
gender CHAR(1),
joining DATE,
birth_date DATE,
PRIMARY KEY (staff_number) );
```

SQL Query to create a table

Python with mongoDB

- Installation:
- We have to install PyMongo interface to work to mongoDB.
- PyMongo is just a driver.

Why MongoDB?

- Flexible schema - supports hierarchical data structure.
- Oriented toward programmers - it supports associative arrays such as php arrays, python dictionaries, JSON objects, Ruby hash etc.
- Lots of MongoDB Drivers and Client Libraries
- Drivers in MongoDB are used for connectivity between client applications and the database. For example, if we have a Python program and we want to connect to MongoDB, then we need to download and integrate the Python driver so that the program can work with the MongoDB database. PyMongo is the driver for Python.
- Flexible deployment.
- Designed for BigData.
- Aggregation Framework.

PyMongo Install

- You can install using following command

$ pip install pymongo

Python with Oracle

Installation:

- Install Oracle
- Install Oracle GUI like sql developer
- Install driver
- Test the connection

Python with Oracle

```
import cx_Oracle
con = cx_Oracle.connect("system", "root1", "localhost/xe")
print("db connected")
cur = con.cursor()
```

Any Queries