DATABASE PROGRAMMING

With Python

Database Programming Overview

Problem:

Write a program in a traditional programming language
 (Java/Python/C++/...) that accesses data in a relational database

Preference:

Support multiple database back-ends with minimal effort

□ Good news:

Relational databases understand SQL

Database Programming Challenges

- SQL standard is implemented in incompatible ways by database vendors
- Each database vendor uses proprietary network protocols and interfaces
- Using SQL to communicate with the database means we have two languages to deal with:
 - Host application language (ex. Python)
 - Database language (SQL)

Database Programming Approaches

- Database-specific vs. Database-independent
- Database API Approaches
 - Embedded SQL
 - Object-Relational Mapping

Python Database Technologies

| | Database Specific | Database Independent |
|---------------------------|---------------------|----------------------|
| Embedded SQL | Python Database API | |
| Object-Relational Mapping | | SQLAIchemy |

Python Database API

- PEP-249 defines standard API for accessing databases using embedded SQL approach
 - <u>https://www.python.org/dev/peps/pep-0249</u>
- □ Many aspects of the standard are database-specific
- Python standard library support SQLite only
- □ Third-party modules implement PEP-249 for other databases

MySQL Python Module

- Implements Python Database API
- Install using pip
 - pip install mysql-connector-python
- - https://dev.mysql.com/doc/connector-python/en
- Class Examples
 - examples/pythondb

Basic Initialization Steps

Import library

- from mysql.connector import connect
- □ Connect to MySQL server
 - con = connect(user='root', password='passw0rd', database='simpledb')
- Obtain cursor object from connection object
 - cursor = con.cursor()
- □ Turn on auto-commit
 - con.autocommit = True
- Connection and cursor should be closed when finished
- See MySQL library doc for details on handling connection errors:
 - <u>https://dev.mysql.com/doc/connector-python/en/connector-python-example-connecting.html</u>

Interacting with Database

Invoke methods on cursor object to execute SQL and obtain results

Insert/update/delete statement example:

```
cursor.execute("""
```

```
update Product
```

```
set Quantity = Quantity + 1
where ProdName = 'Fifi'
```

""")

```
Example: python-mysql-demo.py
```

Retrieving Data

 Select statement example: cursor.execute(""" select ProdId, ProdName from product """)

```
result
[(1, 'Fifi'),
(2, 'Onions'),
(3, 'Potatoes')]
```

```
result = cursor.fetchall()
```



```
for row in result:
```

```
(prodId, prodName) = row # Extract prodId and prodName from row tuple
print(f"{prodId}: {prodName}")
```

Using Program Variables in SQL

Technique #1: String concatenation, format(), or f-strings

```
qty = 5
desc = 'Toothpaste'
cursor.execute(f"""
    update Product
    set Quantity = Quantity + {qty}
    where ProdName = '{desc}'
"""")
```

Must properly quote values using SQL quoting rules
 Unsafe: Possible SQL Injection Vulnerability

Using Program Variables in SQL

□ Technique #2: Pass a tuple of variable values to cursor.execute()

```
qty = 5
desc = 'Toothpaste'
cursor.execute("""
update Product
set Quantity = Quantity + %s
where ProdName = %s
""", (qty, desc))
```

- Use %s markers in query to refer to values in program variables passed in separate tuple
 - Database substitutes variables in place of %s markers in query
 - Do not use quotes around %s in the query
- Safer than Technique #1: SQL injection not possible

SQL Injection

- Scenario: Need to execute queries that search for data specified by the user
- Problem: User should not be allowed to modify the query in undesirable ways
- Demo: examples/pythondb/dbwebapp.py

SQL Injection Vulnerability

- SQL Injection occurs when a user's input is inserted into an SQL query and modifies the behavior of the query in an undesirable way
 - May cause a database error
 - May return more results than desired
 - May modify data or schema in ways the developer did not intend

Preventing SQL Injection

- Two approaches
- Option 1: Carefully validate the user's input. Reject any input that would modify the SQL in an undesirable way.
- Option 2: Use the two-argument form of execute() that passes variables as a separate tuple parameter.
 - This makes it impossible for the user to modify the SQL.

Object-Relational Mapping

- "Object-Relational Impedance Mismatch"
 - Refers to the difficulties programmers encounter when they attempt to store data in program objects in a relational database
- Object-Relational Mapping Libraries address this problem:
 - Programmer defines a class for each table in the database
 - O-R API maps rows in tables to objects in program
- Example:
 - SQLAlchemy library
 - See webapp-sqlalchemy-demo.py