Database Programming

CSCI 220: Database Management and Systems Design

Practice Quiz: SQL DML

- Write the following as SQL queries:
  - List the names of all the boats
  - List names of the sailors along with the names of all the boats they have ever reserved
  - List each sailor's ID and name, along with the number of reservations they have made

Boats

bid	**bname**	**color**
101	Ariel	blue
102	Comet	red
103	Hornet	yellow
104	Lightning	yellow

Reservations

bid	**sid**	**day**
101	22	9/27/2021
102	33	9/28/2021
103	44	9/27/2021
104	44	9/6/2021

Sailors

sid	**sname**	**rating**	**age**
22	Dustin	7	45
33	Lubber	8	55
44	Sally	10	35

Today you will learn…

- How to interact with a database using a general-purpose programming language (e.g., Python)

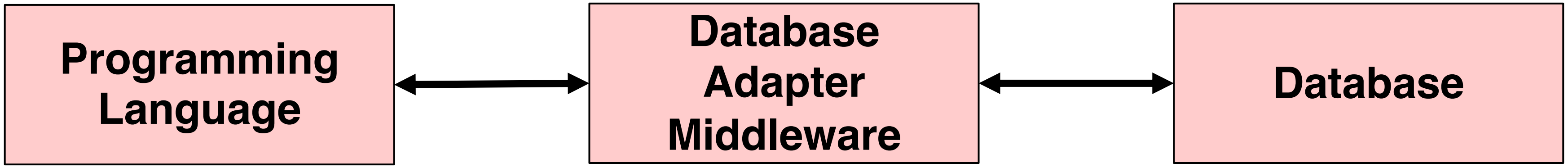
SQL from other Programming Languages

- SQL is not a general-purpose programming language
  - Tailored for data retrieval and manipulation
  - Relatively easy to optimize and parallelize
  - Can't write entire apps in SQL alone
- Options:
  - Make the query language "Turing complete"
    - Avoids the "impedance mismatch" but the language would become more complex
  - **Better idea:** allow SQL to be used from general-purpose programming languages

Dynamic SQL

- Establish a connection to the database
  - With SQLite, just specify the database file name
  - With PostgreSQL, MySQL, etc., specify hostname, username, password
- Use the connection to instantiate a “cursor”
- Use the cursor to:
  - Execute queries
  - Retrieve the results, usually one row at a time
  - Remember to “commit” changes to the DB, so they will persist!
- When finished, close the cursor and connection

Architecture



Programming Language

Database Adapter Middleware

Database

Python	sqlite3 module	sqlite3 DB
Python	psycopg2 module	PostgreSQL DB
Java	JDBC to ODBC	Excel

Database Adapter Middleware

- Application code uses middleware to communicate with the database
  - Send queries to DB
  - Retrieve records from DB
- Middleware and database versions must match
  - Middleware and DB are "tightly coupled"
- Middleware abstracts details of the database from your application
  - You should be able to update your middleware and DB to their latest versions without breaking your application

SQL IN PYTHON

Review: Tuples

- Tuples are similar to lists, but they are immutable
  - Items in a tuple cannot be changed
- Often used to represent parameters to queries, and rows from results

```
>>> alpha = "a","b","c"
>>> alpha = ("a","b","c")
>>> print(alpha)
('a', 'b', 'c')
>>> print(alpha[0])
a
>>> a, b, c = alpha
>>> print(a)
a
>>> numbers = (1, 2, 3)
>>> numbers = (1, )
>>> print(numbers)
(1,)
```

Database API

- Python defines a standard API (objects and methods) for interacting with databases
  - 3rd party developers can write their own libraries which conforms to the standard.
- We will use:
  - The sqlite3 module, which is part of the Python distribution
  - The psycopg2 module, which is available from pypi

Creating a Connection

- A Connection object represents a connection to the database

```
import sqlite3
con = sqlite3.connect('market.db')
```

```
import psycopg2
con = psycopg2.connect(
    dbname="django",
    user="django",
    password="secret",
    host="db.example.com",
    port="5432",
)
```

Getting a Cursor

- A Cursor object is used to execute transactions (via SQL) against the database

```
import sqlite3
con = sqlite3.connect('market.db')
cur = con.cursor()
```

Executing a SQL Statement

- Use the Cursor object’s execute method to run an SQL statement against the database.

```
import sqlite3
con = sqlite3.connect('market.db')
cur = con.cursor()
cur.execute("SELECT * FROM stocks")
print(cur.fetchall())

# Prints
[('APPL', 1000), ('MSFT', 900), …]
```

Close the Connection

- Best practice to close the connection to the database
  - Unclosed connections aren’t usually problem for a local SQLite DB with a single user, but can cause problems for a multi-user DBs

```
import sqlite3
con = sqlite3.connect('market.db')
cur = con.cursor()
cur.execute("SELECT * FROM stocks")
print(cur.fetchall())
con.close()
```

Processing Results

- After calling the `cursor.execute()` method, we can process/interpret the results

- SELECT queries:
  - results will be zero or more rows of data returned from the database

- INSERT, UPDATE, and DELETE queries:
  - the result will be the number of rows (zero or more) affected by the change

Processing SELECT Results

- Save memory by loading one row into memory at a time (or a batch of rows)

```
import sqlite3
con = sqlite3.connect('market.db')
cur = con.cursor()
cur.execute("SELECT * FROM stocks")

# Loads all rows into memory at once
for row in cur.fetchall():
    print(row)

# Loads one row into memory at a time
for row in cur:
    print(row)
```

Processing SELECT Results

- Improve readability by unpacking tuples in your loops

```
import sqlite3
con = sqlite3.connect('market.db')
cur = con.cursor()
cur.execute("SELECT symbol, price FROM stocks")

for symbol, price in cur:
    print(f"{symbol} costs {price}")
```

Processing INSERT/UPDATE/DELETE Results

- The cursor's rowcount attribute is an integer, the number of rows affected.

```
import sqlite3
con = sqlite3.connect('market.db')
cur = con.cursor()
cur.execute(
    "DELETE FROM stocks WHERE symbol ='MSFT'")
print(f"Deleting {cur.rowcount} rows")
```

Committing Changes

- For INSERT, UPDATE, and DELETE queries, you need to call the Connection's commit() method for your changes to persist
  - You can check if the rowcount is what you expect
  - If your program crashes partway through, you won't make an incomplete set of changes (i.e., atomicity)

```
import sqlite3
con = sqlite3.connect('market.db')
cur = con.cursor()
cur.execute(
    "DELETE FROM stocks WHERE symbol ='MSFT'")
con.commit()
```

Problem: SQL Injection

- Most likely, SQL queries in an application will be dependent on some data input by the user.
  - Unless you are careful, your application may be vulnerable to SQL injection – a major security risk
- Vulnerable code:

```
import sqlite3
con = sqlite3.connect('market.db')
cur = con.cursor()
symbol = input("Enter a stock symbol: ")

cur.execute(
       f"SELECT price FROM stocks WHERE symbol='{symbol}'")
# Or
cur.execute("SELECT price FROM stocks WHERE symbol='"
       + symbol + "'")
# Or any time you simply concatenate strings
```

Problem: SQL Injection

SQL injection exploits the syntax of SQL to chain extra statements to an SQL query.

Everything is okay if the user inputs:

```
MSFT
```

But suppose user inputs:

```
MSFT';DROP TABLE stocks AND 't'='t
```

The resulting SQL becomes:

```
SELECT price from stocks
WHERE symbol='MSFT';DROP TABLE stocks AND 't'='t'
```

Problem: SQL Injection

- Should you worry about SQL injection, and other web attacks?
    - YES!

- Bots will automatically test for vulnerabilities in any internet-connected web server

Solution: Parameterized SQL

- Have the database driver, not Python, include your parameters in the query
  - The database knows how to "escape" characters like ' to prevent SQL injection

```
import sqlite3
con = sqlite3.connect('market.db')
cur = con.cursor()
symbol = input("Enter a stock symbol: ")
cur.execute(
    "SELECT price FROM stocks WHERE symbol=?",
    (symbol,))
```

Solution: Parameterized SQL

- Parameterized SQL should be used every time a variable is included in a SQL statement

```
cursor.execute(
    "INSERT INTO stocks VALUES (?,?,?,?,?)",
    (symbol, name, price, earnings, yield))
```

Best Solution

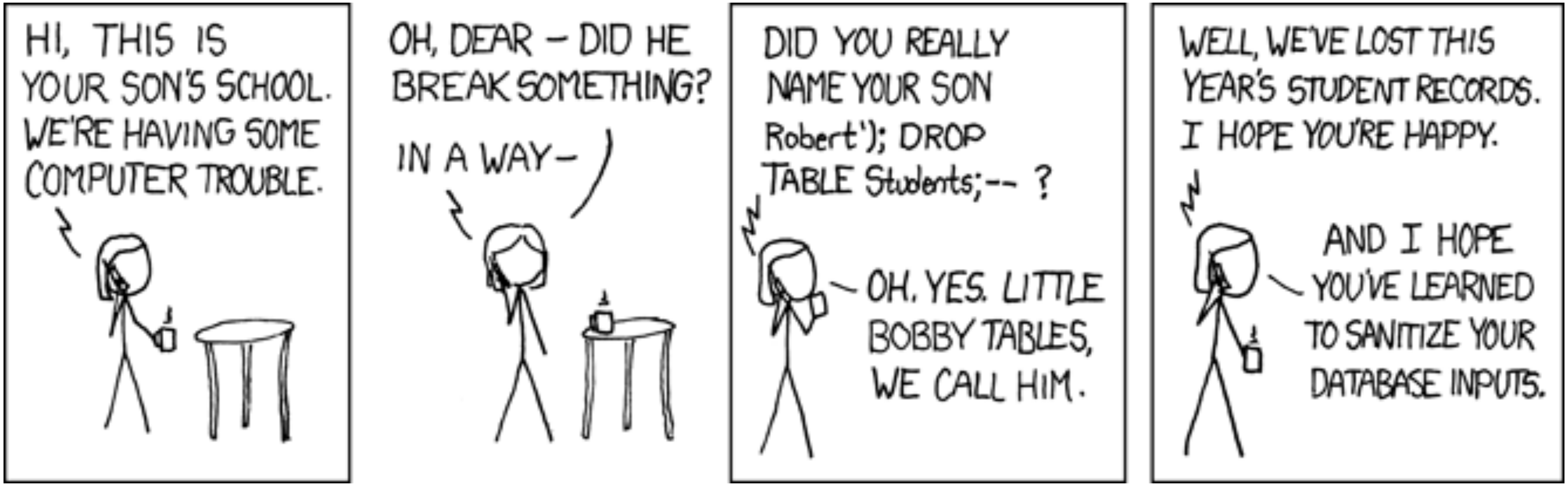
- Use a high-level framework that protects against injection vulnerabilities by default
  - Without protections by default, you are liable to forget – just one mistake can be enough to get hacked

```
symbol = input("Enter a stock symbol: ")

# Safe, and easy!
Stock.objects.get(symbol=symbol)

# Unsafe, but more difficult
Stock.objects.raw(
       f"SELECT * FROM market_stock WHERE symbol={symbol}")
```

SQL Injection


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Source: xkcd.com

Sanitize Your Inputs?