Notes on SQL Programming and Injection Attack

Introduction to Databases

CompSci 316 Fall 2020

DUKE
COMPUTER SCIENCE

Not included in the final exam,
But important for your project

- You have been using SQL programming for your class projects
- This is to discuss "SQL Injection Attack" and "sanitizing inputs" briefly that your code should adhere to
  - Some systems automatically take care of such attacks
  - Remember to mention in your final project report how you handled SQL Injection Attack
    - If you used "Prepared" statement as an optimization, mention it too

Working with SQL through an API

- E.g.: Python psycopg2, JDBC, ODBC (C/C++/VB)
  - All based on the SQL/CLI (Call-Level Interface) standard

- The application program sends SQL commands to the DBMS at runtime
- Responses/results are converted to objects in the application program

Example API: Python psycopg2

```
import psycopg2
conn = psycopg2.connect(dbname='beers')
cur = conn.cursor()
# list all drinkers:
cur.execute('SELECT * FROM Drinker')
for drinker, address in cur:
    print(drinker + ' lives at ' + address)
# print menu for bars whose name contains “a”:
cur.execute('SELECT * FROM Serves WHERE bar LIKE %s', ('%a%',))
for bar, beer, price in cur:
    print('{} serves {} at ${:,.2f}'.format(bar, beer, price))
cur.close()
conn.close()
```

You can iterate over cur one tuple at a time

Placeholder for query parameter

Tuple of parameter values, one for each %s

(note that the trailing “,” is needed when the tuple contains only one value)

More psycopg2 examples

```
# “commit” each change immediately—need to set this option just once at the start of the session
conn.set_session(autocommit=True)
# ...

bar = input('Enter the bar to update: ').strip()

beer = input('Enter the beer to update: ').strip()

price = float(input('Enter the new price: '))

try:

   cur.execute('''
UPDATE Serves
SET price    = %s
WHERE bar = %s AND beer = %s''', (price, bar, beer))

   if cur.rowcount != 1:

      print('{} row(s) updated: correct bar/beer?'\
          .format(cur.rowcount))

except Exception as e:

   print(e)
```

# of tuples modified

Exceptions can be thrown
(e.g., if positive-price constraint is violated)

Prepared statements: motivation

```
while True:
    # Input bar, beer, price...
    cur.execute('''
UPDATE Serves
SET price = %s
WHERE bar = %s AND beer = %s''', (price, bar, beer))
    # Check result...
```

- Every time we send an SQL string to the DBMS, it must perform parsing, semantic analysis, optimization, compilation, and finally execution
- A typical application issues many queries with a small number of patterns (with different parameter values)
- Can we reduce this overhead?

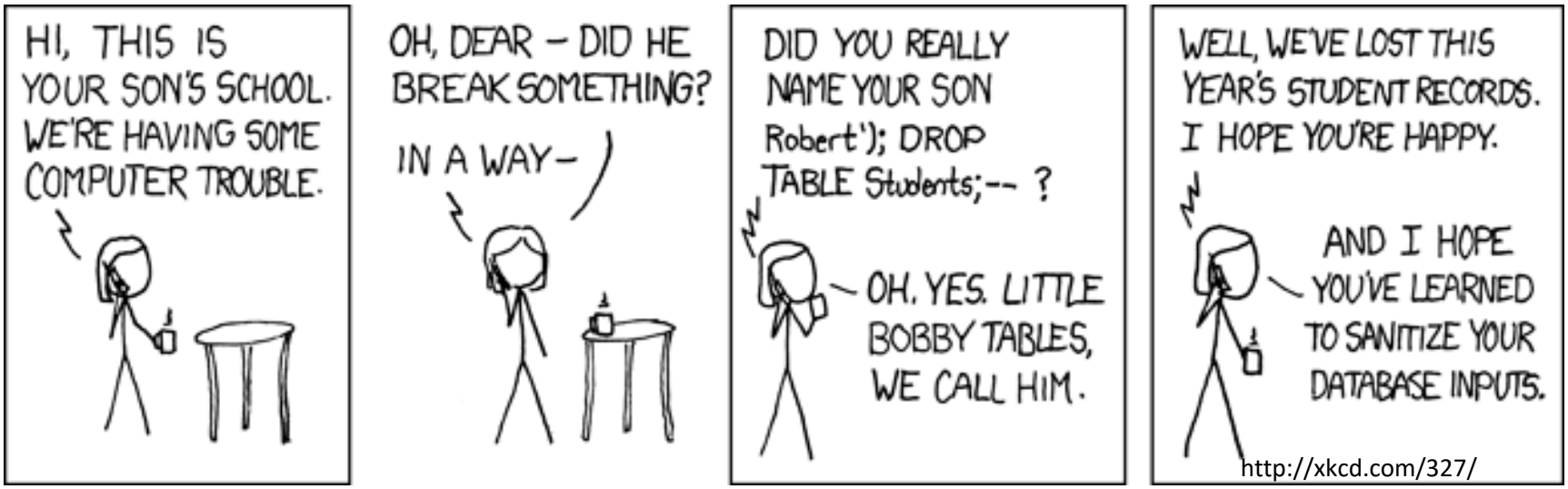
Prepared statements: example

See /opt/dbcourse/examples/psycopg2/ on your VM for a complete code example

```
cur.execute('''
PREPARE update_price AS
UPDATE Serves
SET price = $1
WHERE bar = $2 AND beer = $3''') # Prepare once (in SQL).
                                 # Name the prepared plan,
                                 # and note the $1, $2, ... notation for
                                 # parameter placeholders.
while True:
    # Input bar, beer, price...
    cur.execute('EXECUTE update_price(%s, %s, %s)',\
            (price, bar, beer))  # Execute many times.
                                 # Note the switch back to %s for parameter placeholders.
    # Check result...
```

- The DBMS performs parsing, semantic analysis, optimization, and compilation only once, when it "prepares" the statement
- At execution time, the DBMS only needs to check parameter types and validate the compiled plan
- Most other API's have better support for prepared statements than psycopg2
  - E.g., they would provide a cur.prepare() method

“Exploits of a mom”



- The school probably had something like:

```
cur.execute("SELECT * FROM Students " + \
        "WHERE (name = '" + name + "')")
```

  where name is a string input by user

- Called an SQL injection attack

Guarding against SQL injection

- Escape certain characters in a user input string, to ensure that it remains a single string
  - E.g., ', which would terminate a string in SQL, must be replaced by '' (two single quotes in a row) within the input string
- Luckily, most API's provide ways to "sanitize" input automatically (if you use them properly)
  - E.g., pass parameter values in psycopg2 through %s's

If one fails to learn the lesson…

… P.S. To Ashley Madison's Development Team: You should be embarrased [sic] for your train wreck of a database (and obviously security), not sanitizing your phone numbers to your database is completely amateur, it's as if the entire site was made by Comp Sci 1XX students.

— Creators of CheckAshleyMadison.com

https://en.wikipedia.org/wiki/Ashley_Madison
http://www.washingtonpost.com/news/the-intersect/wp/2015/08/19/how-to-see-if-you-or-your-spouse-appear-in-the-ashley-madison-leak/