Lecture 14:

- Dynamic SQL (cont)

Announcements:

- HW-5 out (due next Thurs)
- Q6 Thurs (dynamic SQL topics)
C++ Example using MySQL Connector

```cpp
#include <iostream>
#include <mysql_connection.h>
#include <cppconn/driver.h>
#include <cppconn/statement.h>
#include <cppconn/exception.h>

int main()
{
    try {
        // connection info
        std::string hst = "cps-database.gonzaga.edu";
        std::string dat = "bowersDB";
        std::string usr = "user";
        std::string pwd = "password";

        // create connection
        sql::Driver* driver = get_driver_instance();
        sql::Connection* cn = driver->connect(hst, usr, pwd);
        cn->setSchema(dat);

        // create statement
        sql::Statement* st = cn->createStatement();
        string q = "SELECT * FROM pet ORDER BY name";

        // execute query
        sql::ResultSet* rs = st->executeQuery(q);

        // print result
        while (rs->next())
            std::cout << "name = " << rs->getString("name") << std::endl;

        // clean up
        delete rs;
        delete st;
        delete cn;
    }
    catch (sql::SQLException &e) {
        std::cout << e.what() << std::endl;
    }
}
```

To compile and run:  ... assuming running on linux

```bash
g++ -I /usr/include/cppconn my_query.cpp -lmysqlcppconn -o my_query
./my_query
```

Note: More examples in the MySQL Connector documentation
Python Example using MySQL Connector

```python
import mysql.connector as mc

def main():
    try:
        # connecton info
        hst = 'cps-database.gonzaga.edu'
dat = 'bowersDB'
usr = 'user'
pwd = 'password'

        # create connection
        cn = mc.connect(host=hst, database=dat, user=usr, password=pwd)

        # result set (called a cursor here)
        rs = cn.cursor()
        q = 'SELECT * FROM pet ORDER BY name'

        # execute the query
        rs.execute(q)

        # display results
        for row in rs:
            print(f'name = {row[1]}')  # name is 2nd attribute

        # clean up
        rs.close()
cn.close()

    except mc.Error as err:
        print(err)

    if __name__ == '__main__':
        main()
```

Note: close() functions should go in a “finally” block

To Install: easiest way is to use pip3

```
w/ pip: pip3 install mysql-connector-python
conda: conda install mysql-connector-python
```

Note: your setup may vary slightly
Handling user input

In most applications user input is used to generate queries:

- e.g., login information, form data, keyword searches

Typically, results in a dynamically created query string

- which can cause issues with “**SQL Injection**” attacks ...
- note similar issues in web-dev with “**Cross-Site Scripting**” (XSS) attacks

Consider the query:

```python
qs = "SELECT password " + 
    "FROM user_table " + 
    "WHERE user_name = '' + usr_input + '');
```

But instead of a username, what if the following is entered?

' OR true; --

Results in the SQL query:

```sql
SELECT password 
FROM user_table 
WHERE user_name = '' OR true; -- '
```

*Note:* You should not store passwords in plain text:

- either don’t store them at all
- or if you must, salt and hash them first
The standard solution is to “quote” input strings

- replace single quote with \' and double quote with \"

Typically handled automatically by libraries when using SQL prepared statements

- the prepared statement allows for string parameters
- the libraries handle the proper “quoting” of input strings

Prepared statements in java (similar in C++, see examples):

```java
String q = "SELECT id, name FROM pet WHERE type = ? AND size = ?";
PreparedStatement st = cn.prepareStatement(q);
st.setString(1, userInput1);
st.setString(2, userInput2);
ResultSet rs = st.executeQuery();
```

- the specific set method depends on datatype

Note: You must use prepared statements with user input in this class!

- ... you must not just concat together query string and user input
User input in Python: ... parameterized strings

Use (older-style) string formatting syntax:

```python
rs = cn.cursor()
q = 'SELECT name, breed FROM pet WHERE type = %s AND size = %s'
rs.execute(q, (user_input_1, user_input_2))  # tuple syntax
```

- for number values, use '%d' instead of '%s'

If you only have one parameter, then:

```python
rs = cn.cursor()
q = 'SELECT name, breed FROM pet WHERE type = %s'
rs.execute(q, (user_input_1,))  # 1-element tuple syntax
```

Note: newer Python format strings are still vulnerable to SQL injection
Updates in Dynamic SQL

For insert, in Java use prepared statements: ... similar in C++

```java
// insert prepared statement
q = "INSERT INTO pet VALUES (?, ?, ?, ?, ?, ?, ?)";
st = cn.prepareStatement(q);

// set values
st.setInt(1, 6);
st.setString(2, "snoopy");
st.setString(3, "dog");
st.setString(4, "beagle");
st.setString(5, "1965-01-01");
st.setString(6, "M");
st.setString(7, "happy");

// execute update
st.execute();
```

- where `set`'s first argument is the parameter (1 for first, etc.)
- see examples for handling date values in Java

In Python ...

```python
# insert example:
q = "INSERT INTO pet VALUES (%s, %s, %s, %s, %s, %s, %s)";
rs = cn.cursor()
rs.execute(q, ('snoopy', 'dog', 'beagle', '1965-01-01', 'M', 'happy'))

# have to commit the change
cn.commit()
```

Additional examples are provided in the example code for HW-5

- including SQL statements for inserting, updating, removing rows
- and for each of the three languages (Java, C++, Python)