Lecture 10:
  • Dynamic SQL (cont)

Announcements:
  • HW-2 out (should be almost done)
  • HW-3 out
  • Moved exam 1 up one day (Tues, Oct 18)

Java and JDBC: A Simple Example

```java
import java.sql.*;

public class MySQLQuery {
    public static void main(String[] args) throws Exception {
        String url = "jdbc:mysql://cps-database.gonzaga.edu/bowersDB";
        Connection cn = DriverManager.getConnection(url, "user", "password");
        Statement st = con.createStatement();
        String query = "SELECT * FROM pet ORDER BY name";
        ResultSet rs = st.executeQuery(query);
        while(rs.next()) {
            String name = rs.getString("name");
            System.out.println("name = " + name);
        }
        rs.close();
        st.close();
        cn.close();
    }
}
```

*Note: examples take shortcuts to fit on a slide
  • no comments, exception handling (try-catch blocks), error checking, etc.*
Java and JDBC: A Simple Example (cont)

To compile and run the program

$ javac MySQLQuery.java
$ java MySQLQuery
name = babe
name = bill
name = fido
name = hobbes
name = toto

Above works on ada ... on your system will need to include the jar:

javac MySQLQuery.java
java -cp .:mysql-connector-java-8.0.30.jar MySQLQuery

Note the MySQL connected can be downloaded from:

other bindings: https://dev.mysql.com/downloads/ (e.g., for C++)

S. Bowers CPSC 321, Fall 2022 3

C++ Example using MySQL Connector

```cpp
#include <iostream>
#include <mysql_connection.h>
#include <driver.h>
#include <connection.h>
#include <resultset.h>
#include <exception.h>

int main()
{
    try {
        sql::Driver* driver = get_driver_instance();
        sql::Connection* cn = driver->connect(HOST, USER, PASS);
        cn->setSchema("bowersDB");
        sql::Statement* st = cn->createStatement();
        string q = "SELECT * FROM pet ORDER BY name";
        sql::ResultSet* rs = stmt->executeQuery(q);
        while (rs->next())
        {
            std::cout << "name = " << rs->getString("name") << std::endl;
            delete rs;
        }
    }
    catch (sql::SQLException &e) {
        std::cout << e.what() << std::endl;
    }
}
```

S. Bowers CPSC 321, Fall 2022 4
C++ Example using MySQL Connector (cont)

To compile:

```
g++ -I /usr/include/cppconn my_query.cpp -l mysqlcppconn -o my_query
```

And run:

```
./my_query
```

More examples in the MySQL Connector documentation

- e.g., creating prepared statements, handling exceptions, types of result sets, etc.

Python Example using MySQL Connector

```python
import mysql.connector as mc
def main():
    try:
        cn = mc.connect(host='cps-database.gonzaga.edu',
                        user='user', password='pass', database='cpsc321')
        rs = cn.cursor()
        q = 'SELECT * FROM pet ORDER BY name'
        rs.execute(q)
        for row in rs:
            print('name = ', row[1])  # name is 2nd attribute
        rs.close()
        cn.close()
    except mc.Error as err:
        print(err
    if __name__ == '__main__':
        main()
```

Note: in examples, close() functions should go in a “finally” block
**Python Example using MySQL Connector**

**To Install:** easiest way is to use pip3

```
pip3 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:

```
SELECT password FROM user_table WHERE user_name = '' OR true; -- ';
```
Handling User Input (cont)

The standard solution is to “quote” input strings

- Replace single quote with \\
and double quote with \\

Typically handled automatically by libraries via “prepared statements”

- a parameterized query that is “prepared” in advance by the DBMS
- the libraries handle the proper “quoting” of input strings

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

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

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

In Python (also see examples from class):

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

If you only have one parameter, then:

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

Additional examples are provided in the example code for HW-3

- including SQL statements for inserting and update
- and for each of the three languages (java, c++, python)