Today …

• Quiz 8
• Dynamic SQL

Homework

• HW 8 due
• HW 9 out
Java and JDBC

Java Database Connectivity (JDBC) API

- A Java API for accessing RDBMSs (RDBMS independent)
- Each specific DBMS implements a JDBC “driver” (i.e., the API)
- Similar to MS ODBC
- Many languages today have DBMS-specific libraries
- ... but most follow JDBC/ODBC style

Must install the specific JDBC driver for your system

- Connector/J for MySQL
- This is just a jar file ...
- The JDBC API comes standard in Java (but not the Driver)
A (super) simple example

```java
import java.sql.*;

public class MyQuery {
    public static void main(String[] args) throws Exception {
        // create connection
        String url = "jdbc:mysql://cps-database.gonzaga.edu/cpsc321";
        String user = "user";
        String pswd = "password";
        Connection cn = DriverManager.getConnection(url, user, pswd);
        // create a statement and execute it
        Statement st = con.createStatement();
        String query = "SELECT * FROM category ORDER BY name";
        ResultSet rs = st.executeQuery(query);
        // print result
        while (rs.next()) {
            String name = rs.getString("name");
            System.out.println("name = " + name);
        }
        rs.close();
        st.close();
        cn.close();
    }
}
```

To compile and run the program

```
$ javac MyQuery.java
$ java MyQuery
name = Action
name = Animation
name = Children
...
```

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

```java
java -cp .:mysql-connector-java-8.0.18.jar MyQuery
```

---

1These examples take shortcuts to fit on one page, e.g., ommitting or simplifying try-catch blocks, etc.
### C++ Example using MySQL Connector

```cpp
#include <iostream>
#include <mysql_connection.h>
#include <driver.h>
#include <connection.h>
#include <resultset.h>
#include <exception.h>
using namespace std;

int main()
{
    try {
        sql::Driver* driver = get_driver_instance();
        sql::Connection* con = driver->connect(HOST, USER, PASS);
        con->setSchema("cpsc321");

        sql::Statement* stmt = con->createStatement();
        string q = "SELECT * FROM category ORDER BY name";
        sql::ResultSet* res = stmt->executeQuery(q);
        while (res->next()) {
            string name = res->getString("name");
            cout << "name = " << name << endl;
        }
        delete res;
        delete stmt;
        delete con;
    }
    catch (sql::SQLException &e) {
        cout << e.what() << endl;
    }
    return 0;
}
```
To compile:

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

And run:

```
./a.out
```

Lots of examples in the MySQL documentation

- For example to create prepared statements, handle exceptions, use various types of cursors, etc.
import mysql.connector

def main():
    try:
        # create a connection
        con = mysql.connector.connect(host='cps-database.gonzaga.edu',
                                        user='user', password='pass',
                                        database='cpsc321')

        # create a result set
        rs = con.cursor()
        q = 'SELECT * FROM category ORDER BY name'
        rs.execute(q)

        # print results
        for row in rs:
            print 'name =', row[1]  # category name is 2nd attribute

        rs.close()
        con.close()

    except mysql.connector.Error as err:
        print err

if __name__ == '__main__':
    main()

Note: in all of these examples, close() functions should go in finally block
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 ...

Simple example of SQL Injection ...

```
// get user input, store in "name" variable
...

// generate query string
String q = "SELECT * FROM user WHERE user_name = " + name + "';";

// execute the query
...
```

What can go wrong ...

Q: Instead of a normal username, what if the following is entered? (MySQL)

```
' OR true; --
```

- Results in the SQL query:

```
"SELECT * FROM user_table WHERE user_name = ' ' OR true; -- ';
```

- This is the “standard” example – but can lead to all kinds of issues

The standard solution is to “quote” input strings

- Replace single quote with \' and double quote with "
Queries with string parameters via prepared statements

In **Java** (also see examples from class):

```java
// create and execute a prepared statement
String q =
    "SELECT COUNT(*) as total FROM film_category " +
    "JOIN film USING (film_id) " +
    "WHERE category_id = ? AND INSTR(title, ?)";
PreparedStatement stmt = con.prepareStatement(q);
stmt.setString(1, userInput1);
stmt.setString(2, userInput2.toUpperCase());
ResultSet rs = stmt.executeQuery();
```

In **C++** (also see examples from class):

```cpp
// create and execute query
string q =
    "SELECT COUNT(*) as total "
    "FROM film_category JOIN film USING (film_id) "
    "WHERE category_id = ? AND INSTR(title, ?)";
sql::PreparedStatement* prep_stmt = con->prepareStatement(q);
prep_stmt->setInt(1, user_input_1);
prep_stmt->setString(2, to_upper(user_input_2));
```

In **Python** (also see examples from class):

```python
# create and execute query
rs = con.cursor()
query = """SELECT COUNT(*) as total
    FROM film_category JOIN film USING (film_id)
    WHERE category_id = %s AND INSTR(title, %s)"""
rs.execute(query, (user_input_1, user_input_2.upper()))
```