Today

• Dynamic SQL

Assignments

• PROJ 2 due
• HW 10 out, PROJ 3 out
Using SQL within a Program

Most of the world’s SQL is generated from software apps!

- Typically through a set of libraries (modules, packages)
- Statements can refer to "host variables" ... regular program variables passed as arguments to queries
- Sometimes called a "Prepared Statement"

SQL query results can be large

- Most languages cannot efficiently (or practically) hold large query results
- To help, DBMSs support "Result Sets"
  - Similar to I/O stream APIs
  - And to "iterators" ... like hasNext(), next() in Java

Different approaches for using SQL within a program

"Dynamic" SQL ... fairly common, what we’ll discuss
  - SQL statements as strings, passed to API calls

Object-Relational Mappings (ORMs) ... also common (e.g., Hibernate)
  - automatic mapping to/from tables and class objects

"Embedded" SQL ... not as common today
  - preprocessor for SQL “embedded” in the programming language
Result Sets

Features vary from one DBMS to another

- Move forward only
- Move forward or backward one row at a time
- Move to arbitrary locations
- Modify or delete records that have been retrieved

Q: What are some advantages/disadvantages?

Result Set placement

- Placed BEFORE the first row of the result
- Getting the next result (logically) “pulls” the value from the DBMS
- Some notification of when last record is obtained

Our plan ...

- Briefly look at how this works in three different languages
- Start with Java and JDBC
- Then look at Python and C++
- Then more examples

\footnote{might not be implemented this way}
Java and JDBC

Java Database Connectivity (JDBC) API

- A Java API for accessing RDBMSs (RDBMS independent)
- Each specific DBMS implements a JDBC “driver”
- That is, implements the API for their system
- Very similar to ODBC
- Many languages today have DBMS-specific libraries, but many follow the JDBC/ODBC API 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

import java.sql.*;

public class MyQuery {

    public static void main(String[] args) throws Exception {
        // create connection
        String url = "jdbc:mysql://147.222.163.1:3306/cpsc321_DB";
        String user = "user";
        String pass = "password";
        Connection cn = DriverManager.getConnection(url, user, pass);
        // create a statement and execute it
        Statement st = cn.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 execute the program

$ javac MyQuery.java
$ java -cp MyQuery
name = Action
name = Animation
name = Children
name = Comedy
...
C++ 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_DB");

        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:

```bash
g++ -I/usr/include/cppconn dbtest.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='147.222.163.1', user='user',
                                       password='pass',
                                       database='cpsc321_DB')

        # 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