Accessing DB from programming languages

JDBC and ODBC

- API (application-program interface) for a program to interact with a database server
- Application makes calls to
  - Connect with the database server
  - Send SQL commands to the database server
  - Fetch tuples of result one-by-one into program variables
- ODBC (Open Database Connectivity) works with C, C++, C#, and Visual Basic
  - Other API's such as ADO.NET sit on top of ODBC
- JDBC (Java Database Connectivity) works with Java

JDBC

- **JDBC** is a Java API for communicating with database systems supporting SQL.
- JDBC supports a variety of features for querying and updating data, and for retrieving query results.
- JDBC also supports metadata retrieval, such as querying about relations present in the database and the names and types of relation attributes.
- Model for communicating with the database:
  - Open a connection
  - Create a “statement” object
  - Execute queries using the Statement object to send queries and fetch results
  - Exception mechanism to handle errors

JDBC Code

```
public static void JDBCexample(String dbid, String userid, String passwd)
{
    try {
        Class.forName ("oracle.jdbc.driver.OracleDriver");
        Connection conn = DriverManager.getConnection(
              "jdbc:oracle:thin:@db.yale.edu:2000:univdb", userid, passwd);
        Statement stmt = conn.createStatement();
           … Do Actual Work ….
        stmt.close();
        conn.close();
    }
    catch (SQLException sqle) {
        System.out.println("SQLException : " + sqle);
    }
}
```

JDBC Code (Cont.)

- Update to database

```
try {
    stmt.executeUpdate(
        "insert into instructor values('77987', 'Kim', 'Physics',
98000)");
} catch (SQLException sqle)
{
   System.out.println("Could not insert tuple. " + sqle);
}
```

- Execute query and fetch and print results

```
    ResultSet rset = stmt.executeQuery(
                              "select dept_name, avg (salary)
                               from instructor
                               group by dept_name");
    while (rset.next()) {
        System.out.println(rset.getString("dept_name") + " " +
                                        rset.getFloat(2));
    }
```

JDBC Code Details

- Getting result fields:
  - **rs.getString("dept_name") and rs.getString(1) equivalent if dept_name is the first argument of select result.**
- Dealing with Null values
  - **int a = rs.getInt("a");**

    if (rs.wasNull()) Systems.out.println("Got null value");

Prepared Statement

- PreparedStatement pStmt = conn.prepareStatement(
  "insert into instructor values(?,?,?,?)");
  pStmt.setString(1, "88877");
  pStmt.setString(2, "Perry");
  pStmt.setString(3, "Finance");
  pStmt.setInt(4, 125000);
  pStmt.executeUpdate();
  pStmt.setString(1, "88878");
  pStmt.executeUpdate();
- WARNING: always use prepared statements when taking an input from the user and adding it to a query
  - NEVER create a query by concatenating strings
  - "insert into instructor values(' " + ID + " ', ' " + name + " ', " + " ' + dept name + " ', " ' balance + ")“
  - What if name is “D’Souza”?

SQL Injection

- Suppose query is constructed using
  - "select * from instructor where name = '" + name + "'"
- Suppose the user, instead of entering a name, enters:
  - X' or 'Y' = 'Y
- then the resulting statement becomes:
  - "select * from instructor where name = '" + "X' or 'Y' = 'Y" + "'"
  - which is:
    - select * from instructor where name = 'X' or 'Y' = 'Y'
  - User could have even used
    - X'; update instructor set salary = salary + 10000; --
- Prepared statement internally uses:
  "select * from instructor where name = 'X\' or \'Y\' = \'Y'
  - **Always use prepared statements, with user inputs as parameters**

Metadata Features

- ResultSet metadata
- E.g., after executing query to get a ResultSet rs:
  - ResultSetMetaData rsmd = rs.getMetaData();

```
for(int i = 1; i <= rsmd.getColumnCount(); i++) {
    System.out.println(rsmd.getColumnName(i));
        System.out.println(rsmd.getColumnTypeName(i));
}
```

- How is this useful?

ODBC

- Open DataBase Connectivity(ODBC) standard
  - standard for application program to communicate with a database server.
  - application program interface (API) to
    - open a connection with a database,
    - send queries and updates,
    - get back results.
- Applications such as GUI, spreadsheets, etc. can use ODBC

ODBC (Cont.)

- Each database system supporting ODBC provides a "driver" library that must be linked with the client program.
- When client program makes an ODBC API call, the code in the library communicates with the server to carry out the requested action, and fetch results.
- ODBC program first allocates an SQL environment, then a database connection handle.
- Opens database connection using SQLConnect(). Parameters for SQLConnect:
  - connection handle,
  - the server to which to connect
  - the user identifier,
  - password
- Must also specify types of arguments:
  - SQL_NTS denotes previous argument is a null-terminated string.

ODBC Code

```
int ODBCexample()
{
  RETCODE error;
  HENV    env;     /* environment */
  HDBC    conn;  /* database connection */
  SQLAllocEnv(&env);
  SQLAllocConnect(env, &conn);
  SQLConnect(conn, “db.yale.edu", SQL_NTS, "avi", SQL_NTS,
   "avipasswd", SQL_NTS);
  { …. Do actual work … }

  SQLDisconnect(conn);
  SQLFreeConnect(conn);
  SQLFreeEnv(env);
 }
```

ODBC Code (Cont.)

- Program sends SQL commands to the database by using SQLExecDirect
- Result tuples are fetched using SQLFetch()
- SQLBindCol() binds C language variables to attributes of the query result
  - When a tuple is fetched, its attribute values are automatically stored in corresponding C variables.
  - Arguments to SQLBindCol()
    - ODBC stmt variable, attribute position in query result
    - The type conversion from SQL to C.
    - The address of the variable.
    - For variable-length types like character arrays,
      - The maximum length of the variable
      - Location to store actual length when a tuple is fetched.
      - Note: A negative value returned for the length field indicates null value
- Good programming requires checking results of every function call for errors; we have omitted most checks for brevity.

ODBC Code (Cont.)

- Main body of program

```
  char deptname[80];
float salary;
int lenOut1, lenOut2;
HSTMT stmt;
char * sqlquery = "select dept_name, sum (salary)
                   from instructor
                   group by dept_name";
SQLAllocStmt(conn, &stmt);
error = SQLExecDirect(stmt, sqlquery, SQL NTS);
if (error == SQL SUCCESS) {
      SQLBindCol(stmt, 1, SQL C CHAR, deptname , 80, &lenOut1);
      SQLBindCol(stmt, 2, SQL C FLOAT, &salary, 0 , &lenOut2);
      while (SQLFetch(stmt) == SQL SUCCESS) {
            printf (" %s %g\n", deptname, salary);
      }
}
SQLFreeStmt(stmt, SQL DROP);
```

ODBC Prepared Statements

- **Prepared Statement**
  - SQL statement prepared: compiled at the database
  - Can have placeholders: E.g. insert into account values(?,?,?)
  - Repeatedly executed with actual values for the placeholders
- To prepare a statement
  SQLPrepare(stmt, <SQL String>);
- To bind parameters
  SQLBindParameter(stmt, <parameter#>,
  … type information and value omitted for simplicity..)
- To execute the statement
  retcode = SQLExecute(stmt);
- To avoid SQL injection security risk, do not create SQL strings directly using user input; instead use prepared statements to bind user inputs

More ODBC Features

- **Metadata features**
  - finding all the relations in the database and
  - finding the names and types of columns of a query result or a relation in the database.
- By default, each SQL statement is treated as a separate transaction that is committed automatically.
  - Can turn off automatic commit on a connection
    - SQLSetConnectOption(conn, SQL_AUTOCOMMIT, 0)}
  - Transactions must then be committed or rolled back explicitly by
    - SQLTransact(conn, SQL_COMMIT) or
    - SQLTransact(conn, SQL_ROLLBACK)

Perl DBI

```
use DBI;

my $dbh = DBI->connect("dbi:Sybase:server=SERV;packetSize=8092",
   "anonymous", "anonymous");
if (! defined $dbh) {
die ("Cannot connect to server\n");
}

my $mysqlqry = <STDIN>;

$dbh->do("set textsize 65535");

my $qh = $dbh->prepare($mysqlqry) || die ("Cannot prepare\n");
$qh->execute() || die ("Cannot execute\n");

while (my @row = $qh->fetchrow()){
processrow($row);
}
```

Other resources

- See website for JDBC information that works on Glue machines
- PHP documentation:
  http://www.php.net/manual/en/index.php
  http://us3.php.net/mysql
- Ruby on Rails
  http://api.rubyonrails.org/

- Last year's class:
  http://www.cs.umd.edu/class/fall2009/cmsc424/project_resources.html