Database Systems
Lecture07 – Advanced SQL

Beomseok Nam (남범석)
bnam@skku.edu

Accessing SQL from a Programming Language

- A database programmer must have access to a general-purpose programming language for at least two reasons
- Not all queries can be expressed in SQL, since SQL does not provide the full expressive power of a general-purpose language.
- Non-declarative actions -- such as printing a report, interacting with a user, or sending the results of a query to a graphical user interface -- cannot be done from within SQL.

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
- 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:
  - [1] Open a connection
  - [2] Create a “Statement” object
  - [3] Execute queries using the “Statement” object to send queries and fetch results
  - [4] Exception mechanism to handle errors

JDBC Code

```
public static void JDBCexample(String dbid, String userid,
                               String passwd)
{
    try {
       Class.forName ("com.mysql.jdbc.Driver");
       Connection conn = DriverManager.getConnection(
             "jdbc:mysql://localhost:3306/db_name",
              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
  - `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();
```

- For queries, use pStmt.executeQuery(), which returns a ResultSet

- 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 which you get as inputs**
  - "insert into instructor values(' " + ID + " ', ' " + name + " ', " + " ' + dept name + " ', " ' balance + ")“

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));
}
```

Metadata (Cont)

- DatabaseMetaData
  - provides methods to get metadata about database
- DatabaseMetaData dbmd = conn.getMetaData();

```
ResultSet rs = dbmd.getColumns(null, "univdb", "department", "%");
// Returns: One row for each column; row has a number of attributes
// such as COLUMN_NAME, TYPE_NAME
while( rs.next()) {
        System.out.println(rs.getString("COLUMN_NAME"),
                           rs.getString("TYPE_NAME");
   }
```

Transaction Control in JDBC

- By default, each SQL statement is treated as a separate transaction that is committed automatically
  - bad idea for transactions with multiple updates
- Can turn off automatic commit on a connection
  - `conn.setAutoCommit(false);`
- Transactions must then be committed or rolled back explicitly
  - `conn.commit();` or
  - `conn.rollback();`
- `conn.setAutoCommit(true)` turns on automatic commit.

Other JDBC Features

- Calling functions and procedures
  - Oracle PL/SQL, MS TransactSQL, etc
  - `CallableStatement cStmt1 = conn.prepareCall("{? = call some function(?)}");`
  - `CallableStatement cStmt2 = conn.prepareCall("{call some procedure(?,?)}");`
- Handling large object types
  - `getBlob()` and `getClob()` that are similar to the `getString()` method, but return objects of type `Blob` and `Clob`, respectively
  - get data from these objects by `getBytes()`
  - associate an open stream with Java Blob or Clob object to update large objects
    - `blob.setBlob(int parameterIndex, InputStream inputStream).`

SQLJ

- JDBC is overly dynamic, errors cannot be caught by compiler
- SQLJ: embedded SQL in Java

```
#sql iterator deptInfoIter(String dept_name,
                           int avgSal);
deptInfoIter iter = null;
#sql iter = { select dept_name, avg(salary)
              from instructor
              group by dept name };
while (iter.next()) {
     String deptName = iter.dept_name();
      int avgSal = iter.avgSal();
      System.out.println(deptName + " " + avgSal);
}
iter.close();
```

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
- Was defined originally for Basic and C, versions available for many languages.

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

ODBC Code

```
int ODBCexample()
{
    RETCODE error;
    HENV    env;      /* environment */
    HDBC    conn;     /* database connection */
    SQLAllocEnv(&env);
    SQLAllocConnect(env, &conn);
    SQLConnect(conn, "localhost", SQL_NTS,
        "bnam", SQL_NTS, "changethis", SQL_NTS);
                                  // SQL_NTS: NULL
    { …. Do actual work … }       // Terminated String

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

ODBC Code (Cont.)

- Program sends SQL commands to 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)`

MySQL C API

```
MYSQL *conn = mysql_init(NULL);
mysql_real_connect(conn, "localhost", "bnam",
                   "changethis", "dbbnam", 0, NULL, 0);
char * sqlquery = "select dept_name, avg(salary)\
                from instructor group by dept_name ";
mysql_query(conn, sqlquery);
MYSQL_RES *result = mysql_store_result(conn);
int num_fields = mysql_num_fields(result);
MYSQL_ROW row;
while ((row = mysql_fetch_row(result))) {
     for(int i=0; i<num_fields; i++){
         printf (" %s ", row[i]? row[i] : "NULL");
     }
     printf("\n");
}
mysql_free_result(result);
mysql_close(conn);
```

Embedded SQL

- The SQL standard defines embeddings of SQL in a variety of programming languages such as C, Java, and Cobol.
- A language to which SQL queries are embedded is referred to as a **host language**, and the SQL structures permitted in the host language comprise *embedded* SQL.
- The basic form of these languages follows that of the System R embedding of SQL into PL/I.
- **EXEC SQL** statement is used to identify embedded SQL request to the preprocessor

  EXEC SQL <embedded SQL statement > END_EXEC

  Note: this varies by language (for example, the Java embedding uses # SQL { …. };)

Example Query

- Specify the query in SQL

```
void main ()
{
    EXEC SQL BEGIN DECLARE SECTION;
        int credit_amount;
    EXEC SQL END DECLARE SECTION;

    …
    EXEC SQL
        declare c cursor for
        select ID, name
        from student
        where tot_cred > :credit_amount
    END_EXEC
    …
}
```

- From within a host language, find the ID and name of students who have completed more than the number of credits stored in variable credit_amount.