CS 2550 / Spring 2006
Principles of Database Systems

05 – SQL Programming

Alexandros Labrinidis
University of Pittsburgh

How to program applications

- Using existing languages:
  - Embed SQL into "Host" language
    - ESQL, SQLJ
  - Use a library of functions
    - JDBC
- Design a new language
- **Problem**: impedance mismatch
  - Data types
  - Accessing results in table form

Roadmap

- Embedded SQL
- Dynamic SQL
- ODBC
- JDBC

SQL is not enough

- SQL does not provide the full functionality of general-purpose programming languages
  - less powerful
  - on purpose: SQL can be automatically optimized and executed efficiently
- SQL cannot perform "non-declarative" actions:
  - cannot interact with user
  - cannot print results
  - cannot manage a Graphical User Interface

Embedded SQL

- Solution:
  - Bind together SQL with general purpose programming language
- Programming language = host language
- SQL included within host lang. = embedded SQL (ESQL)
- How:
  - include embedded SQL within the host language
  - run pre-processor before compiling program
- Format:
  - EXEC SQL <embedded SQL statement> END-EXEC

How ESQL/host lang. communicate

- Variables from host language can be included in ESQL
  - Variable **X** is included within SQL as **:X**
- Query results are retrieved one tuple at a time:
  - **Open**()
  - while (**Fetch**())
    - perform action on each result tuple
  - **Close**()
- Must check return codes for errors

ESQL – Cursors

- From within a host language, find the names and cities of customers with more than the **X** dollars in account
- Specify the query in SQL and declare a *cursor* for it
  - A cursor is a "pointer" to a specific tuple within a set of results

```
EXEC SQL
    declare c cursor for
    select  customer_name, customer_city
    from    depositor, customer, account
    where depositor.customer_name = customer.customer_name
            and depositor account_number = account.account_number
             and account.balance > :X
END-EXEC
```

ESQL – Execution

- The **open** statement causes the query to be evaluated
  EXEC SQL **open** *c* END-EXEC
- The **fetch** statement causes the values of one tuple in the query result to be placed on host language variables.
  EXEC SQL **fetch** *c* **into** :*cust_name, :cust_city* END-EXEC
  Repeated calls to **fetch** get successive tuples in the query result
- A variable called SQLSTATE in the SQL communication area (SQLCA) gets set to '02000' to indicate no more data is available
- The **close** statement causes the database system to delete the temporary relation that holds the result of the query.
  EXEC SQL **close** *c* END-EXEC

ESQL – Updates

- Can update tuples fetched by cursor by declaring that the cursor is for update

```
declare c cursor for
   select      *
   from        account
   where       branch-name = 'Perryridge'
 for update
```

- Loop over results using fetch
- To update tuple at the current location of cursor

```
update      account
 set        balance = balance + 100
 where current of c
```

Roadmap

- Embedded SQL
- Dynamic SQL
- ODBC
- JDBC

Dynamic SQL

- Allow programs to construct and submit SQL queries at run-time
  - Embedded SQL = static SQL, queries must be defined before preprocessing/compiling
- Example of dynamic SQL from within a C program.

```
char *  sqlprog =   "update account
                     set      balance = balance * 1.05
                     where    account_number = ?"
EXEC SQL prepare dynprog from :sqlprog;
char account [10] = "A-101";
EXEC SQL execute dynprog using :account;
```

- The dynamic SQL program contains a ?, which is a place holder for a value that is provided when the SQL program is executed.

Dynamic SQL – Execution

- Well-defined Application Program Interface (API)
- General structure of Dynamic SQL:
  - Connect to DB server (new session)
  - Execute statements
    - Prepare
    - Open/fetch/close
    - Updates
  - Commit/Rollback
  - Close session

Roadmap

- Embedded SQL
- Dynamic SQL
- ODBC
- JDBC

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:
  - the connection handle,
  - the server to which to connect
  - the user identifier,
  - the password

Roadmap

- Embedded SQL
- Dynamic SQL
- ODBC
- JDBC

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
  - query about relations present in the database
  - query 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 ← this is different than ODBC

JDBC Code Example

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

JDBC Code – Main Body

- Update database

```
try {
    stmt.executeUpdate(  "insert into account values
                                    ('A-9732', 'Perryridge', 1200)");
} catch (SQLException sqle) {
    System.out.println("Could not insert tuple. " + sqle);
}
```

- Execute query and fetch and print results

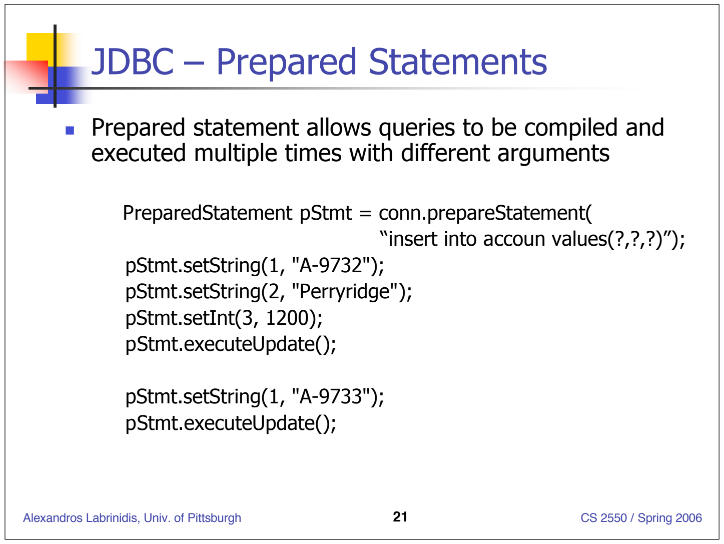
```
ResultSet rset = stmt.executeQuery( "select branch_name, avg(balance)
                                        from account
                                        group by branch_name");
while (rset.next()) {
      System.out.println(
                 rset.getString("branch_name") + "  " + rset.getFloat(2));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

JDBC Code – II

- Getting result fields:
  - rs.getString("branchname") and rs.getString(1) equivalent if branchname 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");
```

- Correct Quotation
  - "insert into account values ('A-9732', …)"



JDBC – Prepared Statements

- Prepared statement allows queries to be compiled and executed multiple times with different arguments

```
PreparedStatement pStmt = conn.prepareStatement(
                                    “insert into accoun values(?,?,?)”);
 pStmt.setString(1, "A-9732");
 pStmt.setString(2, "Perryridge");
 pStmt.setInt(3, 1200);
 pStmt.executeUpdate();

 pStmt.setString(1, "A-9733");
 pStmt.executeUpdate();
```