SQL and Applications

https://midas.bu.edu/classes/CS460/

Writing Applications with SQL

- SQL is not a general purpose programming language.
  - + Tailored for data retrieval and manipulation
  - + Relatively easy to optimize and parallelize
- Awkward to write entire apps in SQL

- Options:
  - Make the query language “Turing complete”
    - Avoids the “impedance mismatch”
    - makes “simple” relational language complex
  - Allow SQL to be embedded in regular programming languages.

Cursors

- Can declare a cursor on a relation or query
- Can open a cursor
- Can repeatedly fetch a tuple (moving the cursor)
- Special return value when all tuples have been retrieved.
- ORDER BY allows control over the order tuples are returned.
  - Fields in ORDER BY clause must also appear in SELECT clause.
- LIMIT controls the number of rows returned (good fit w/ORDER BY)
- Can also modify/delete tuple pointed to by a cursor
  - A “non-relational” way to get a handle to a particular tuple

Database APIs

- A library with database calls (API)
  - special objects/methods
  - passes SQL strings from language, presents result sets in a language-friendly way
  - ODBC a C/C++ standard started on Windows
  - JDBC a Java equivalent
  - Most scripting languages have similar things
  - E.g. in Python there's the "psycopg2" driver
- ODBC/JDBC try to be DBMS-neutral
  - at least try to hide distinctions across different DBMSs

Summary

- Relational model has well-defined query semantics
- SQL provides functionality close to basic relational model
  - (some differences in duplicate handling, null values, set operators, …)
- Typically, many ways to write a query
  - DBMS figures out a fast way to execute a query, regardless of how it is written.

Database Application Development

JDBC

- Part of Java, very easy to use
- Java comes with a JDBC-to-ODBC bridge
  - So JDBC code can talk to any ODBC data source
  - E.g. look in your Windows Control Panel for ODBC drivers!
- JDBC tutorial online
  - http://developer.java.sun.com/developer/Books/JDBCTutorial/

JDBC Basics: Connections

- A Connection is an object representing a login to a database

```
// GET CONNECTION
Connection con;
try {
   con = DriverManager.getConnection(
      "jdbc:odbc:bankDB",
      userName,password);
} catch(Exception e){ System.out.println(e);  }
```

- Eventually you close the connection

```
// CLOSE CONNECTION
try { con.close(); }
catch (Exception e) { System.out.println(e); }
```

JDBC Basics: Statements

- You need a Statement object for each SQL statement

```
// CREATE STATEMENT
Statement stmt;
try {
    stmt = con.createStatement();
} catch (Exception e){
    System.out.println(e);
}
```

Soon we'll say stmt.executeQuery("select …");

JDBC Basics: ResultSet

- A ResultSet object serves as a *cursor* for the statement's results (stmt.executeQuery())

```
// EXECUTE QUERY
ResultSet results;
try {
    results = stmt.executeQuery(
        "select * from branch")
} catch (Exception e){
    System.out.println(e);
}
```

- Obvious handy methods:
  - results.next() advances cursor to next tuple
    - Returns "false" when the cursor slides off the table (beginning or end)
  - "scrollable" cursors:
    - results.previous(), results.relative(int), results.absolute(int), results.first(), results.last(), results.beforeFirst(), results.afterLast()

ResultSet Metadata

- Can find out stuff about the ResultSet schema via ResultSetMetaData

```
ResultSetMetaData rsmd = results.getMetaData();
int numCols = rsmd.getColumnCount();
int i, rowcount = 0;

// get column header info
for (i=1; i <= numCols; i++){
    if (i > 1) buf.append(",");
    buf.append(rsmd.getColumnLabel(i));
}
buf.append("\n");
```

- Other ResultSetMetaData methods:
  - getColumnType(i), isNullable(i), etc.

Updating Current of Cursor

- Update fields in current of cursor:

```
result.next();
result.updateInt(“assets", 10M);
```

- Also updateString, updateFloat, etc.
- Or can always submit a full SQL UPDATE statement
  - Via executeQuery()

- The original statement must have been CONCUR_UPDATABLE in either case!

Cleaning up Neatly

```
try {
  // CLOSE RESULT SET
  results.close();
  // CLOSE STATEMENT
  stmt.close();
  // CLOSE CONNECTION
  con.close();
} catch (Exception e) {
    System.out.println(e);
}
```

Putting it Together (w/o try/catch)

```
Connection con = DriverManager.getConnection("jdbc:odbc:weblog",userName,password);
Statement stmt = con.createStatement();
ResultSet results = stmt.executeQuery("select * from Sailors")
ResultSetMetaData rsmd = results.getMetaData();
int numCols = rsmd.getColumnCount(), i;
StringBuffer buf = new StringBuffer();

while (results.next() && rowcount < 100){
  for (i=1; i <= numCols; i++) {
     if (i > 1) buf.append(",");
     buf.append(results.getString(i));
  }
  buf.append("\n");
```

Similar deal for web scripting langs

- Common scenario today is to have a web client
  - A web form issues a query to the DB
  - Results formatted as HTML
- Many web scripting languages used
  - jsp, asp, PHP, etc.
  - most of these are similar, look a lot like jdbc with HTML mixed in

E.g. PHP/Postgres

```
<?php   $conn = pg_pconnect("dbname=cowbook user=jmh\
                             password=secret");
  if (!$conn) {
    echo "An error occured.\n";
    exit;
  }
  $result = pg_query ($conn, "SELECT * FROM Sailors");
  if (!$result) {
    echo "An error occured.\n";  exit;
  }
  $num = pg_num_rows($result);
  for ($i=0; $i < $num; $i++) {
    $r = pg_fetch_row($result, $i);
    for ($j=0; $j < count($r); $j++) {
        echo "$r[$j] ";
    }
    echo "<BR>";
  }
?>
```