CS 235: Introduction to Databases

Svetlozar Nestorov

Lecture Notes #18

Outline

- Embedded SQL.
- Call-Level Interface (CLI).
- Java Database Connectivity (JDBC).

Embedded SQL

- Standard for combining SQL with a host language.
- SQL statements are converted to procedure calls in the host language by a preprocessor.
- Begin SQL statements with *EXEC SQL*.

Shared Variables

- The interface between SQL and the host language is through shared variables.

```
EXEC SQL BEGIN DECLARE SECTION;
   declarations of shared variables in host
   language syntax
EXEC SQL END DECLARE SECTION;
```

Use of Shared Variables

- In SQL, shared variables are preceded by a colon.
  - Can be used as constants in SQL statements.
  - Can get values from SQL statements and pass values to host language.
- In the host language, shared variables are used as any other variables.

Example

- Look up the price that a given bar charges for a given beer.

```
EXEC SQL BEGIN DECLARE SECTION;
   char aBeer[21], aBar[21];
   float aPrice;
EXEC SQL END DECLARE SECTION;
/* read in the beer and the bar */
EXEC SQL SELECT price
            INTO :aPrice
            FROM Sells
            WHERE beer = :aBeer AND bar = :aBar;
/* print the price */
```

Embedded Queries

- Modification queries.
  - Return no results; can be used anywhere.
- Single-row select queries.
  - Return a single tuple; can be read into shared variables.
- Multiple-row select queries.
  - Return many tuples; can be used with cursors.

Cursors

- Declare a cursor.

  EXEC SQL DECLARE c CURSOR FOR <query>;
- Open a cursor.

  EXEC SQL OPEN c;
- Fetch a tuple.

  EXEC SQL FETCH c INTO <vars>;

Example (1/2)

- Find the prices of all beers sold in Spoon.

```
EXEC SQL BEGIN DECLARE SECTION;
  char aBeer[21];
  float aPrice;
EXEC SQL END DECLARE SECTION;
EXEC SQL DECLARE spoonBeers CURSOR FOR
  SELECT beer, price
  FROM Sells
  WHERE bar = 'Spoon';
```

Example (2/2)

```
EXEC SQL OPEN CURSOR spoonBeers;
while(1) {
  EXEC SQL FETCH spoonBeers
      INTO :aBeer, :aPrice;
  if (NO_MORE_TUPLES) break;
  /* print out the beer and the price */
}
EXEC SQL CLOSE CURSOR spoonBeers;
```

Modifying Base Relations

- A cursor can range over a base relation.

  EXEC SQL DECLARE c CURSOR FOR Sells;
- Modifications can be made only to the current tuple.

```
EXEC SQL DELETE FROM Sells
  WHERE CURRENT OF c;
```

- Any condition can be applied in the host language.

Dynamic SQL

- So far, fixed queries with possibly some parameters.
- What if we want run ad-hoc queries?
- Dynamic SQL
  - Prepare statement (not known at compile time.)
  - Execute statement.

Dynamic SQL Syntax

- Prepare a query.
  EXEC SQL PREPARE <query-name>
  FROM <query>;
- Execute a query.
  EXEC SQL EXECUTE <query-name>;

Example

- Read a query and run it.

```
EXEC SQL BEGIN DECLARE SECTION;
  char query[255];
EXEC SQL END DECLARE SECTION;
while (1)
  /* read query */
  EXEC SQL PREPARE q FROM :query;
  EXEC SQL EXECUTE q;
```

Execute-Immediate

- If the query is to be executed only once the prepare and execute statements can be combined.
  EXEC SQL EXECUTE IMMEDIATE <query>;

SQL/CLI

- Call-Level Interface: call library functions and procedures within a host language.
- Data types:
  - Environments: DBMS installation.
  - Connections: logins to DBMS.
  - Statements: SQL statements.
  - Descriptions: query results or parameters.

Data Type Instances

- Create environment, connection, and statement handles with SQLAllocHandle(T,I,O)
  - T is the type, e.g.SQL_HANDLE_ENV.
  - I is the input handle (higher-level handle):
    - statement < connection < environment
  - O is the output handle.

Example

```
SQLHENV myEnv;
SQLHDBC myCon;
SQLAllocHandle(SQL_HANDLE_ENV,
  SQL_NULL_HANDLE, &myEnv);
SQLAllocHandle(SQL_HANDLE_DBC,
  myEnv,  &myCon);
```

Processing Statements

- Prepare and execute.

SQLPrepare(<statement-handle>,
<statement>,
<length of statement>)

SQLExecute(<statement-handle>)

Example

SQLPrepare(myStmt, “SELECT bar, beer FROM Sells WHERE price < 3.00”, SQL_NTS)

SQLExecute(myStmt)

or

SQLExecDirect(myStmt, “SELECT bar, beer FROM Sells WHERE price < 3.00”, SQL_NTS)

Fetching Tuples

- Every statement has an implied cursor associated with it.
- SQLFetch(<stmt-handle>) returns the next tuple from the result of the executed statement.

Binding Variables

- Before fetching we need to indicate where the tuple attributes should be stored.

SQLBindCol(<stmt-handle>,
<attribute-pos>,
<attribute-type>,
<var-ptr>,
<var-size>,
<var-info-ptr>);

Example

```
SQLExecDirect(myStmt, “SELECT bar, beer
   FROM Sells WHERE price < 3.00”, SQL_NTS);
SQLBindCol(myStmt, 1, SQL_CHAR, &aBar,
   size(aBar), &aBarInfo);
SQLBindCol(myStmt, 2, SQL_CHAR, &aBeer,
   size(aBeer), &aBeerInfo);
while (SQLFetch(myStmt) != SQL_NO_DATA)
{
   /* Cheers! */
}
```

Parameterized Queries

- Bind variables to query parameters, so you can execute the same statement several times with different parameters.

SQLPrepare(myStmt, “INSERT(bar, beer) VALUES(?,?)”, SQL_NTS);

SQLBindParameter(myStmt, 1,…,aBar,…);

SQLBindParameter(myStmt, 2,…,aBeer,…);

SQLExecute(myStmt);

JDBC

- Java Database Connectivity (JDBC)
  - Similar to SQL/CLI and ODBC but adapted to object-oriented Java.
- JDBC drivers are similar to environments in CLI.
  - Platform, implementation, and installation dependent.
- DriverManager object.

JDBC Connection

- Connect with DriverManager by specifying the DBMS URL, username, and password.

```
Connection myCon =
  DriverManager.getConnection(
      <DB URL>, <username>, <psswd>);
```

Statements

- Two types of statements:
  - Statement can accept any string that is an SQL statement and execute it.
  - PreparedStatement has a fix SQL statement.

```
Statement s1 = myCon.createStatement();
PreparedStatement s2 =
  myCon.createStatement(<SQL-stmt>);
```

Executing Statements

- JDBC distinguishes between queries and modifications.
- Both Statement and PreparedStatement have two methods:
  - executeQuery
  - executeUpdate
- For Statement the methods take a parameter.

Example

```
PreparedStatement s2 =
  myCon.createStatement("SELECT
  bar,beer FROM Sells WHERE price <
  3.0");
ResultSet cheapBeers = s2.executeQuery();
Statement s1 = myCon.createStatement();
s1.executeUpdate("INSERT INTO Sells
  Values('Spoon', 'Bud', 3.0)");
```

Accessing Results

- ResultSet class objects are similar to cursors.
- Method next() gets the next tuple.
  - Must be called once to get the first tuple.
  - Returns FALSE when tuples are exausted.

```
cheepBeers.next()
```

Accessing Attributes

- Call an appropriate method, depending on the type of attribute, on the ResultSet object.
  - Position of the attribute is a aprameter
- getInt(i), getString(i), getFloat(i).

Example

```
while (cheepBeers.next()) {
   aBar = cheepBeers.getString(1);
   aBeers = cheepBeers.getString(2);
   /* print out a map to the bar */
}
```

Parameterized Queries

- PreparedStatements can be parameterized
  - Use ? to denote a parameter.
- Use methods setString, setInt, setFloat.
- Then run executeQuery or update.