EMBEDDED SQL

MICHAEL LIUT (LIUTM@MCMASTER.CA)
DEPARTMENT OF COMPUTING AND SOFTWARE
MCMASTER UNIVERSITY

SE 3DB3 Fall 2016

(Slides adapted from Dr. Fei Chiang, Diane Horton, examples from J. Ullman, J. Widom)

Problems with using interactive SQL

- Standard SQL does not allow recursion nor loops.
  - E.g., Two profs are "colleagues" if they've co-taught a course or share a colleague.
  - We can't write a query to find all colleagues of a given professor because we have no loops or recursion.
- You can't control the format of its output.
- You want to run queries that are *based on* user input, not have users writing actual queries.

SQL + a conventional language

- If we can combine SQL with code in a conventional language, we can solve these problems.
- But we have another problem:
  - SQL is based on relations, and conventional languages have no such type.
- It is solved by
  - feeding tuples from SQL to the other language one at a time, and
  - feeding each attribute value into a particular variable.

Approaches

- Three approaches for combining SQL and a general-purpose language:

1. Stored Procedures
2. Statement-level Interface
3. Call-level interface

Three Approaches

1. Stored Procedures

- The SQL standard includes a language for defining "stored procedures", which can
  - have parameters and a return value,
  - use local variables, ifs, loops, etc.,
  - execute SQL queries.
- Stored procedures can be used in these ways:
  - called from the interpreter,
  - called from SQL queries,
  - called from another stored procedure.

Example (just to give you an idea)


```
CREATE FUNCTION BandW(y INT, s CHAR(15))   RETURNS BOOLEAN
IF NOT EXISTS
   (SELECT *
    FROM Movies
    WHERE year = y AND studioName = s)
THEN RETURN TRUE;
ELSIF 1 <=
   (SELECT COUNT(*)
    FROM Movies
    WHERE year = y AND studioName = s AND
          genre = 'comedy')
THEN RETURN TRUE;
ELSE RETURN FALSE;
END IF;
```

Not very standard

- The language is called SQL/PSM (Persistent Stored Modules).
  - It came into the SQL standard in SQL3, 1999.
  - By then, various commercial DBMSs had already defined their own proprietary languages for stored procedures
  - They have generally stuck to them.

2. Statement-level interface (SLI)

- Embed SQL statements into code in a conventional language like C or Java.
  - SQL statements can refer to host variables (including special variables used to return status).
  - Two special "error" variables:
    SQLCODE (long, is negative if an error has occurred)
    SQLSTATE (char[6], predefined codes for common errors)
  - Must include a statement to *connect* to the right database.

Statement Level Interface

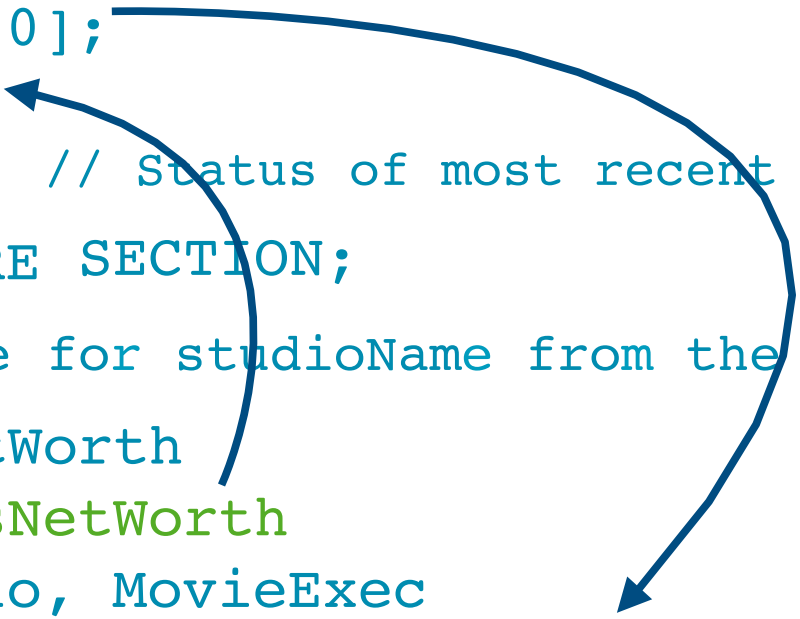
- Use a preprocessor to replace the SQL with calls written in the host language to functions defined in an SQL library.
- Special syntax indicates which bits of code the preprocessor needs to convert.

Constructs

- Language constructs:
  - Connecting to a database:
    EXEC SQL CONNECT
  - Declaring variables:
    EXEC SQL BEGIN (END) DECLARE SECTION
  - Statements:
    EXEC SQL Statement;

Example (just to give you an idea)


```
void printNetWorth() {
  EXEC SQL BEGIN DECLARE SECTION;
    char studioName[50];
    int presNetWorth;
    char SQLSTATE[6]; // Status of most recent SQL stmt
  EXEC SQL END DECLARE SECTION;
  /* OMITTED: Get value for studioName from the user. */
  EXEC SQL SELECT netWorth
           INTO :presNetWorth
           FROM Studio, MovieExec
           WHERE Studio.name = :studioName;
  /* OMITTED: Report back to the user */
```



Cursors

- Can declare a cursor on a relation or query statement (which generates a relation).
- Can *open* a cursor, and repeatedly *fetch* a tuple then *move* the cursor, until all tuples have been retrieved.
  - Can use a special clause, called ORDER BY, in queries that are accessed through a cursor, to control the order in which tuples are returned.
- Can also modify/delete tuple pointed by a cursor.

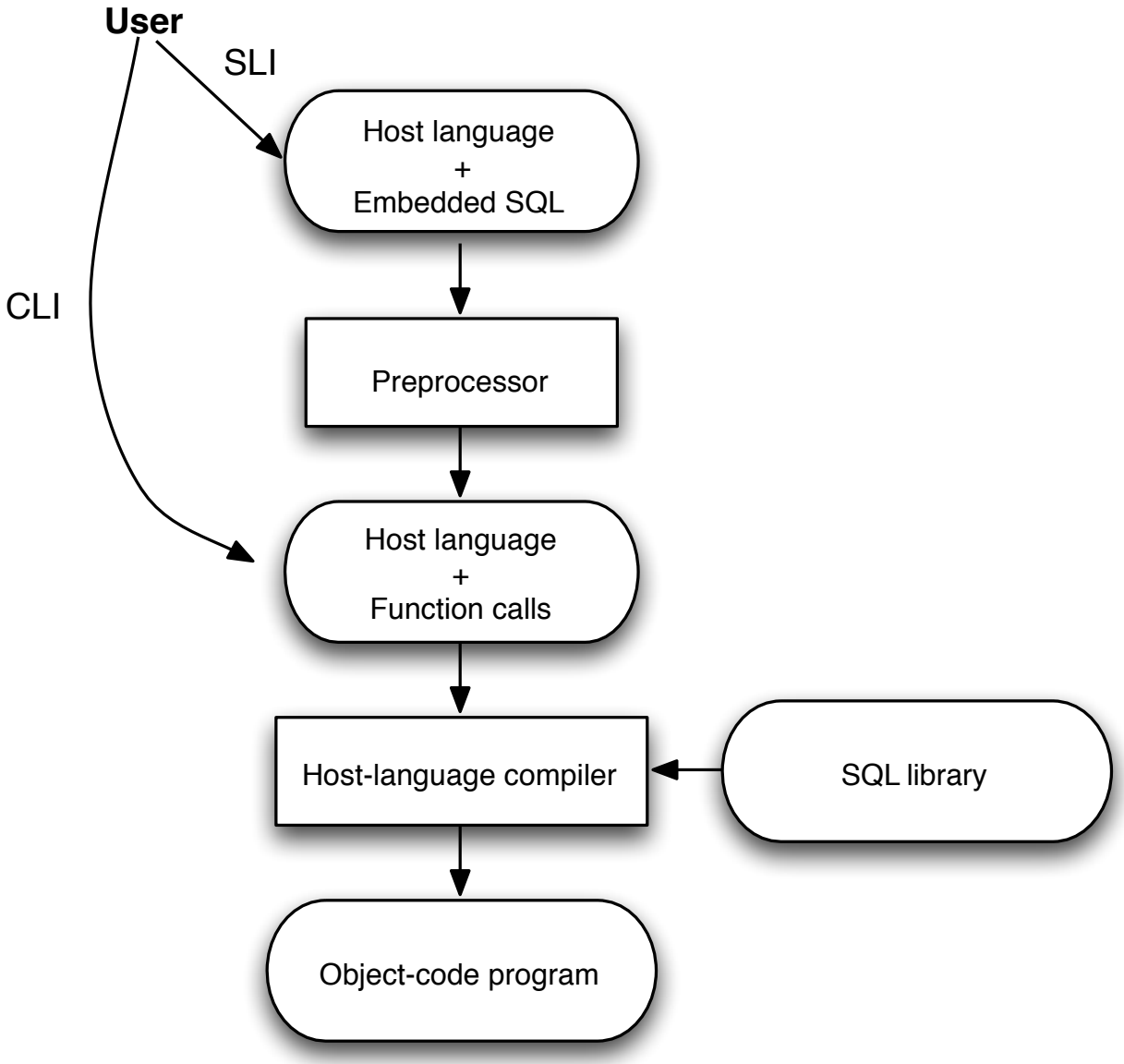
A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

Cursor that gets names of students who've enrolled in a course, in alphabetical order

Students(sname, sid, age, gpa)
Enrolled(sid, cid)

```
EXEC SQL DECLARE sinfo CURSOR FOR
    SELECT  S.sname
    FROM  Students S, Enrolled E
    WHERE  S.sid=E.sid
    ORDER BY  S.sname
```

Big picture


User
SLI
CLI
Host language
+
Embedded SQL
Preprocessor
Host language
+
Function calls
Host-language compiler
SQL library
Object-code program

3. Call-level interface (CLI)

- Instead of using a pre-processor to replace embedded SQL with calls to library functions, write those calls yourself.
- Eliminates need to preprocess.
- Each language has its own set of library functions for this.
  - for C, it's called SQL/CLI
  - for Java, it's called JDBC
  - for PHP, it's called PEAR DB
- We'll look at just one: JDBC.

JDBC

JDBC Example

Do this once in your program:

```
/* Get ready to execute queries. */
import java.sql.*;
/* A static method of the Class class. It loads the
   specified driver */
Class.forName("com.ibm.db2.jcc.DB2Driver");
String url = "jdbc:db2://localhost:5000/exampleDB";
Connection conn = DriverManager.getConnection(
   url, user, pw);
/* Continued ... */
```

Do this once per query in your program:


```
/* Execute a query and iterate through the resulting
   tuples. */
PreparedStatement execStat = conn.prepareStatement(
   "SELECT netWorth FROM MovieExec");

ResultSet worths = execStat.executeQuery();
while (worths.next()) {
   int worth = worths.getInt(1);
   /* If the tuple also had a float and another int
      attribute, you'd get them by calling
      worths.getFloat(2) and worths.getInt(3).
      Or you can look up values by attribute name.
      Example: worths.getInt(netWorth)
   */
   /* OMITTED: Process this net worth */
}
```

Exceptions can occur

- Any of these calls can generate an exception.
- Therefore, they should be inside try/catch blocks.

```
try {
    /* OMITTED: JDBC code */
} catch (SQLException ex) {
    /* OMITTED: Handle the exception */
}
```

- The class `SQLException` has methods to return the `SQLSTATE`, etc.

What is “preparation”?

- Preparing a statement includes parsing the SQL, compiling and optimizing it.
- The resulting `PreparedStatement` can be executed any number of times without having to repeat these steps.

If the query isn't known until run time

- You may need input and computation to determine the query.
- You can hard-code in the parts you know, and use "`?`" as a placeholder for the values you don't know.
- This is enough to allow a `PreparedStatement` to be constructed.
- Once you know values for the placeholders, methods `setString`, `setInt`, etc. let you fill in those values.

Example


```
PreparedStatement studioStat =
    conn.preparedStatement( “INSERT INTO
       Studio(name,address) VALUES(?, ?)”

);

/* OMITTED: Get values for studioName and studioAddr */

studioStat.setString(1, studioName);
studioStat.setString(2, studioAddr);
studioStat.executeUpdate();
```

Why not just build the query in a string?

- We constructed an incomplete `preparedStatement` and filled in the missing values using method calls.
- Instead, we could just build up the query in an ordinary string at run time, and ask to execute that.
- There are classes and methods that will do this in JDBC.
- Disadvantage: it is vulnerable to injections
  - insertion of strings into a query with malicious intent.
- Always use a `preparedStatement` whenever possible.

Example: createStatement( )


```
Statement stat = conn.createStatement();
String query =
  "SELECT networth
   FROM MovieExec
   WHERE execName = 'Spielberg';
  "
ResultSet worths = stat.executeQuery(query);
```

Example: Vulnerable Code

Suppose we want the user to provide the string to compare to

You can do this rather than hard-coding `Spielberg` into the query:

```
Statement stat = conn.createStatement();

String who =/* get a string from the user */

String query=
  "SELECT networth
   FROM MovieExec
   WHERE execName =     '"  + who + "';
  "
ResultSet worths = stat.executeQuery(query);
```

A gentle user does no harm

If a user enters `Eastwood`, the SQL code we execute is this:

```
SELECT networth
FROM MovieExec
WHERE execName = 'Eastwood';
```

Nothing bad happens.

An injection can exploit the vulnerability

What could a malicious user enter?

```
SELECT networth
FROM MovieExec
WHERE execName = '?????????????';
```

Jones' OR '1' = '1

Queries vs updates in JDBC

- The previous examples used `executeQuery`.
- This method is only for queries.
- For SQL statements that change the database (insert, delete or modify tuples, or change the schema), use the analogous method `executeUpdate`.