Embedded SQL

CSC343 Introduction to Databases

UNIVERSITY OF
TORONTO

Problems with using interactive SQL

- Standard SQL is not "Turing-complete".
  - 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.

- Most DB users shouldn't be writing SQL queries!
  - We 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.

- Issue: SQL is based on relations, and conventional languages have no such type.

- Solution:
  - feed tuples from SQL to the other language one at a time;
  - feed each attribute value into a particular variable.

Approaches

- Three approaches for combining SQL and a general-purpose language:
  - Stored Procedures
  - Statement-level Interface
  - Call-level interface

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,
  - be the action that a trigger performs.

Example (just to give an idea)

- A boolean function `Comedy(y INT, s CHAR(15))` that returns true iff
  - movie studio s produced no movies in year y, or
  - produced at least one comedy.

- Reference: Ullman and Widom textbook, chapter 9

Reference: textbook figure 9.1.3

```
CREATE FUNCTION Comedy(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;
```

Calling a stored procedure

-
  ```sql
  SELECT StudioName
  FROM Studios
  WHERE Comedy(2010, StudioName);
  ```

Not standard!

- The language is called SQL/PSM (Persistent Stored Modules).
  - It came into the SQL standard in SQL3, 1999.
- By then, commercial DBMSs had defined their own proprietary languages for stored procedures
  - They have generally stuck to them.
- PostgreSQL has defined PL/pgSQL.
  - It supports some, but not all, of SQL/PSM.
  - Reference: Chapter 39 of the PostgreSQL documentation.

2. Statement-level interface (SLI)

- Embed SQL statements into code in a conventional language like C or Java.
- 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.

Example, in C (just to give you an idea)

Reference: textbook example 9.7

```
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 */
}
```

Data from host language ⇒ SQL

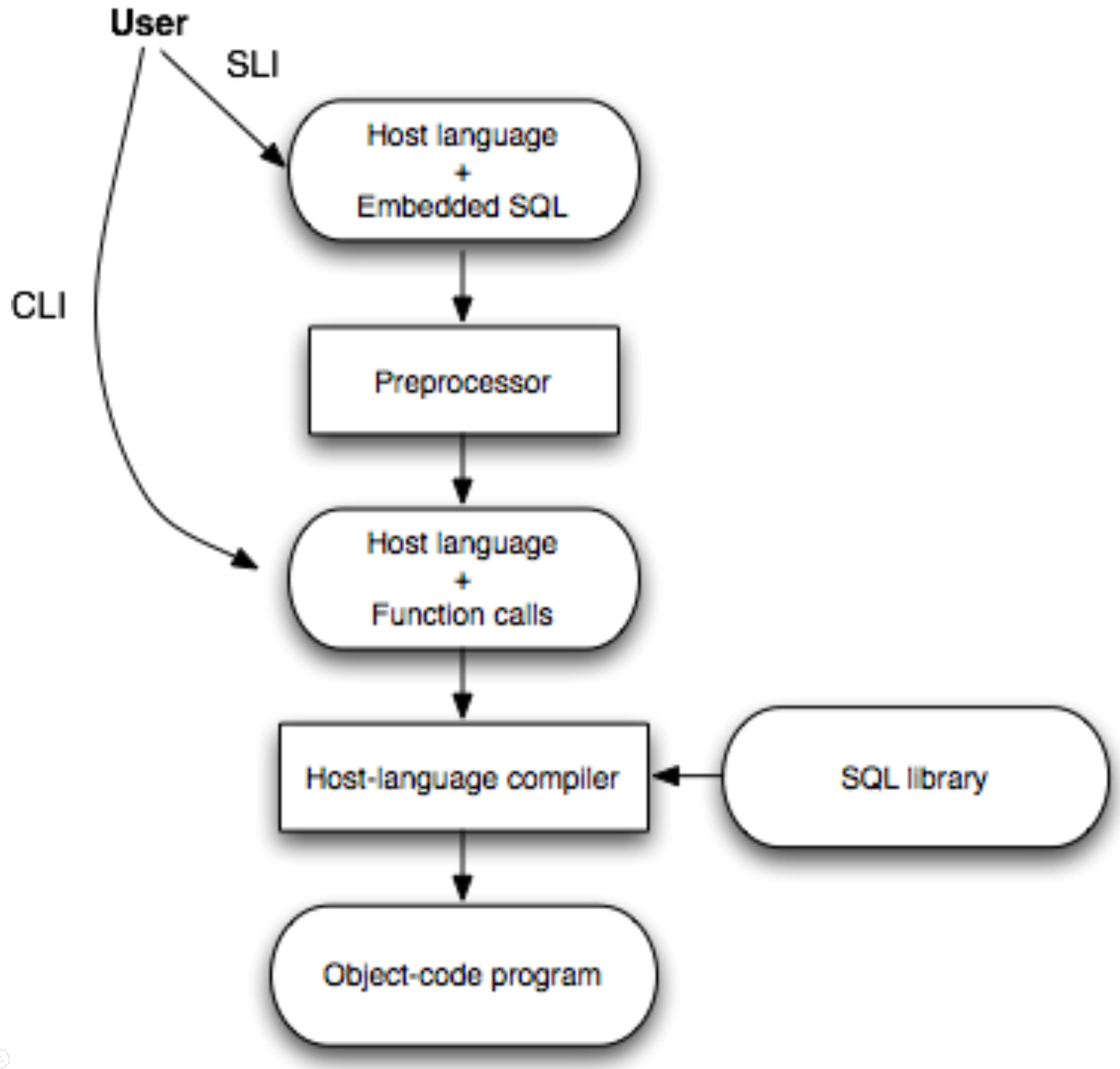
- Some special syntax tells the preprocessor things like this:
  - “Variable `studioName` my C code may be referred to by the SQL that’s embedded in my C code.”

- Uses the keywords `DECLARE SECTION`.

Data from SQL ⇒ host language

- Again, shared variables make the connection.
- If the query will yield a single tuple, an `INTO` clause can be added to the query.
  - It lists the host-language variables that should receive the attribute values from that one tuple.
- For multi-tuple results, a "cursor" allows you to iterate over the tuples.
  - Much like an iterable object in an Object-oriented language.
  - We won't cover this, but see figures 9.8 and 9.9 for examples.

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, we write those calls.

- 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 (textbook section 9.6)

Do this once in a program:

```
/* Get ready to execute queries. */
import java.sql.*;

/* A static method of the Class class. It loads the
   specified driver */
Class.forName(“org.postgresql.jdbc.Driver”);

Connection conn = DriverManager.getConnection(
  jdbc:postgresql://localhost:5432/csc343h-<my user_name>,
  <user_name>,
   “”);
/* Continued ... */
```

The arguments to getConnection

- `jdbc:postgresql`
  We'll use this, but it could be, e.g., `jdbc:mysql`

- `localhost:5432`
  Use this for the CS Teaching Labs.

- `csc343h-userid` and `userid`
  Substitute your userid on the CS Teaching Labs.

- “”
  Password (unrelated to your password).
  Literally use the empty string.

Do this once per query in a 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 */
}
```

The Java details

- For full details on the Java classes and methods used, see the Java API documentation:

  https://docs.oracle.com/javase/8/docs/api/java/sql/package-summary.html

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.

Prepare separately vs execute immediately

- We can combine preparation and execution.
- Separate looked like this:

```
PreparedStatement pStat =
    conn.prepareStatement(
    “SELECT netWorth FROM MovieExec”);
ResultSet worths =
     pStat.executeQuery();
```

- Combined looks like this:

```
Statement stat =
    conn.createStatement();
ResultSet worths =
    stat.executeQuery(
```

What is “preparation”?

- Preparing a statement includes:
  - parsing the SQL
  - compiling
  - optimizing

- 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

- We may need input and computation to determine exactly what the query should be.

- In that case:
    - Hard-code in the parts known.
    - Use the character `?` as a placeholder for the values not known. (Don't put it in quotes!)

- This is enough to allow a `PreparedStatement` to be constructed.

- Once all values for the placeholders are known, use methods `setString`, `setInt`, etc., to fill in those values.

Example (Figure 9.22)

```
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.

Example that builds the query in a string

- We can just use a `Statement`, and give it a String to execute.

```
// stat cannot be compiled & optimized (yet).
Statement stat = conn.createStatement();


String query =
  “SELECT networth
   FROM MovieExec
   WHERE execName like ‘%Spielberg%’;
  ”

// executeQuery can now compile, optimize and run
// the query.
ResultSet worths = stat.executeQuery(query);
```

What could go wrong?

Example: vulnerable code

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

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

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

String query =
  “SELECT networth
   FROM MovieExec
   WHERE execName like ‘%” + who + “%’;
  ”

ResultSet worths = stat.executeQuery(query);
```

If a user enters `Milch`, the SQL code executed is this:

```
SELECT networth
FROM MovieExec
WHERE execName like ‘%Milch%’;
```

Nothing bad happens.

An injection can exploit the vulnerability

What could a malicious user enter?

```
SELECT networth
FROM MovieExec
WHERE execName like ‘%?????????????%’;
```

An injection can exploit the vulnerability

But if a malicious user enters

```
   Milch%’; drop table Contracts; --
```

the code we execute is this:

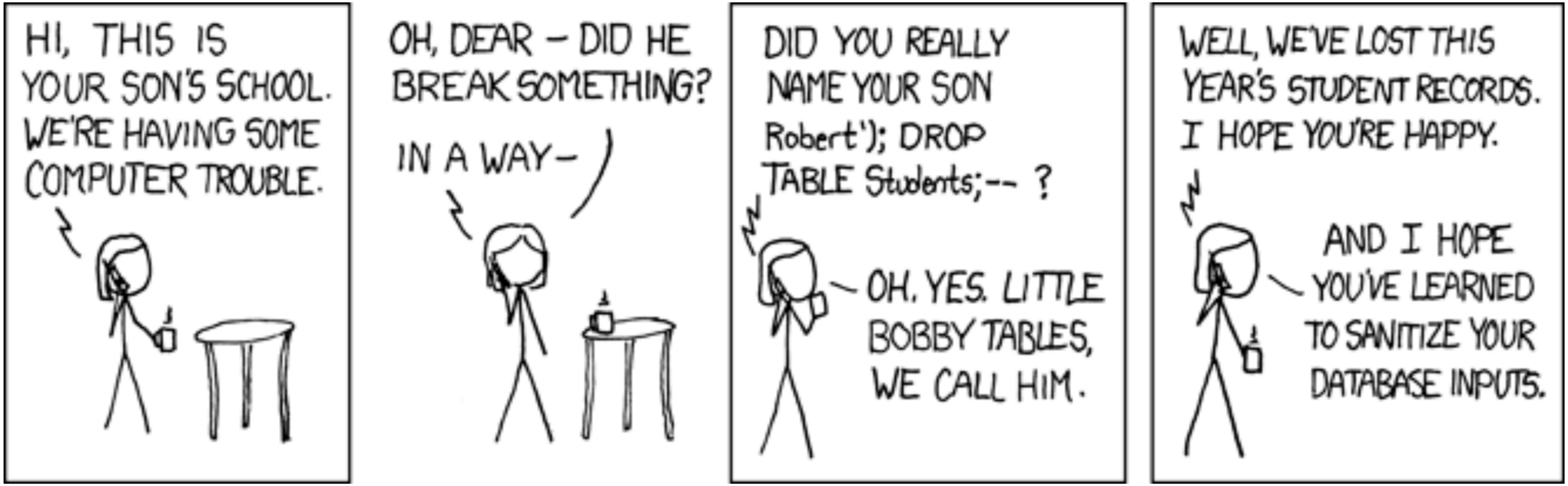
```
SELECT networth
FROM MovieExec
WHERE execName like ‘%Milch%’; DROP TABLE Contracts; --%’;
```

In other words:

```
SELECT networth
FROM MovieExec
WHERE execName like ‘%Milch%’;

DROP TABLE Contracts; --%’;
```

Ouch!


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Reference: https://xkcd.com/327/

Always use a `PreparedStatement`

- This was an example of an injection.
- The simple approach of giving a String to a `Statement` is vulnerable to injections.
- Moral of the story: always use a `PreparedStatement`.

Queries vs updates in JDBC

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