Embedded SQL

Author: Diane Horton
with examples from Ullman and Widom

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.
- You can't control the format of its output.
- And most users shouldn't be writing SQL queries!
  - 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:
  - A SQL query yields a table, 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

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.

Example Stored Procedure

- A boolean function `QuietYear(y INT, s CHAR(15))` that returns true iff
  - movie studio s produced no movies in year y, or
  - produced at most 10 comedies.

- Reference: Ullman and Widom textbook, chapter 9

Reference: textbook figure 9.1.3 (example slightly modified here)

```
CREATE FUNCTION QuietYear(y INT, s CHAR(15)) RETURNS BOOLEAN
IF NOT EXISTS
    (SELECT *
     FROM Movies
     WHERE year = y AND studioName = s)
THEN RETURN TRUE;
ELSIF 10 <=
    (SELECT COUNT(*)
     FROM Movies
     WHERE year = y AND studioName = s AND
           genre = 'comedy')
THEN RETURN TRUE;
ELSE RETURN FALSE;
END IF;
```

Using a stored procedure

Once defined, a stored procedure 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.

Calling the Stored Procedure in a query

```
SELECT StudioName
FROM Studios
WHERE QuietYear(2010, StudioName);
```

Not very standard

- The language is called SQL/PSM (Persistent Stored Modules).
    - It came into the SQL standard in SQL3, 1999.
    - Reference: textbook, section 9.4
- 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 the standard.
    - Reference: Chapter 43 of the PostgreSQL documentation.

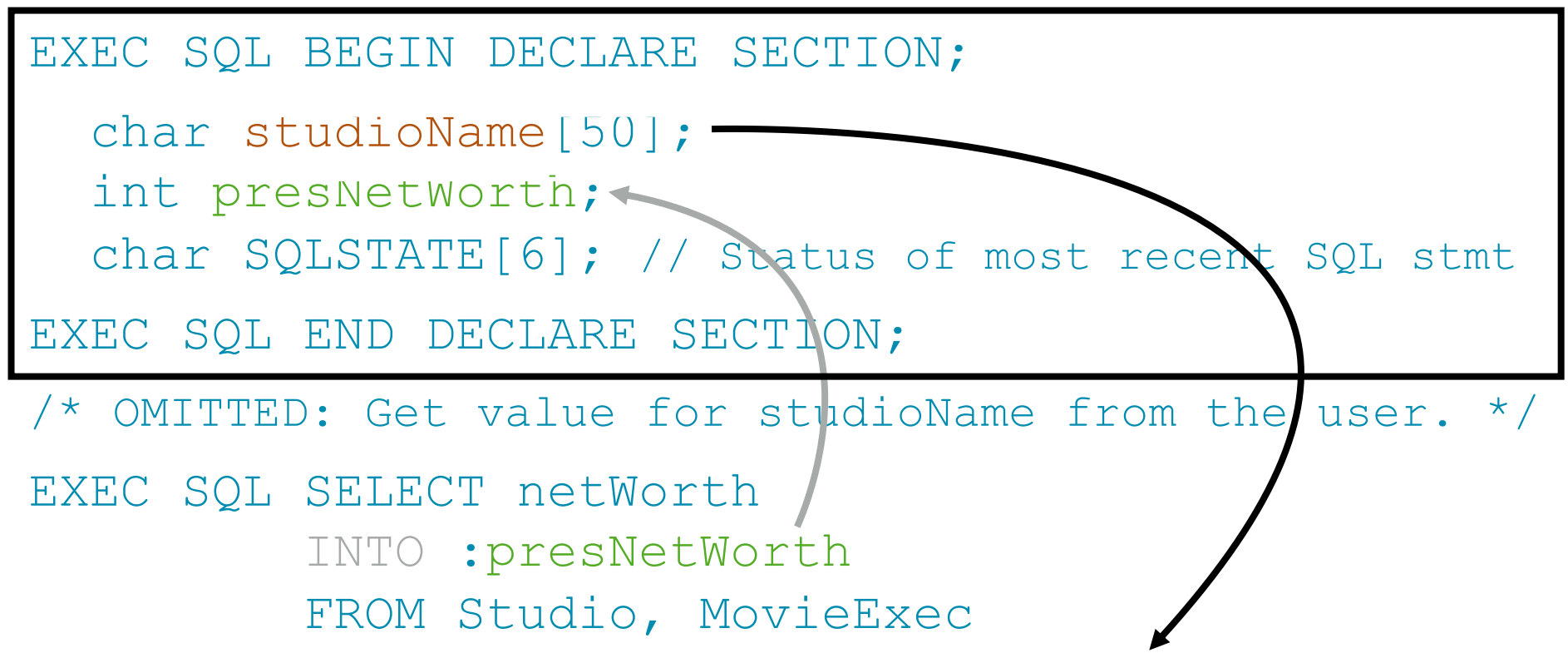
2. Statement-level interface (SLI)

- 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)

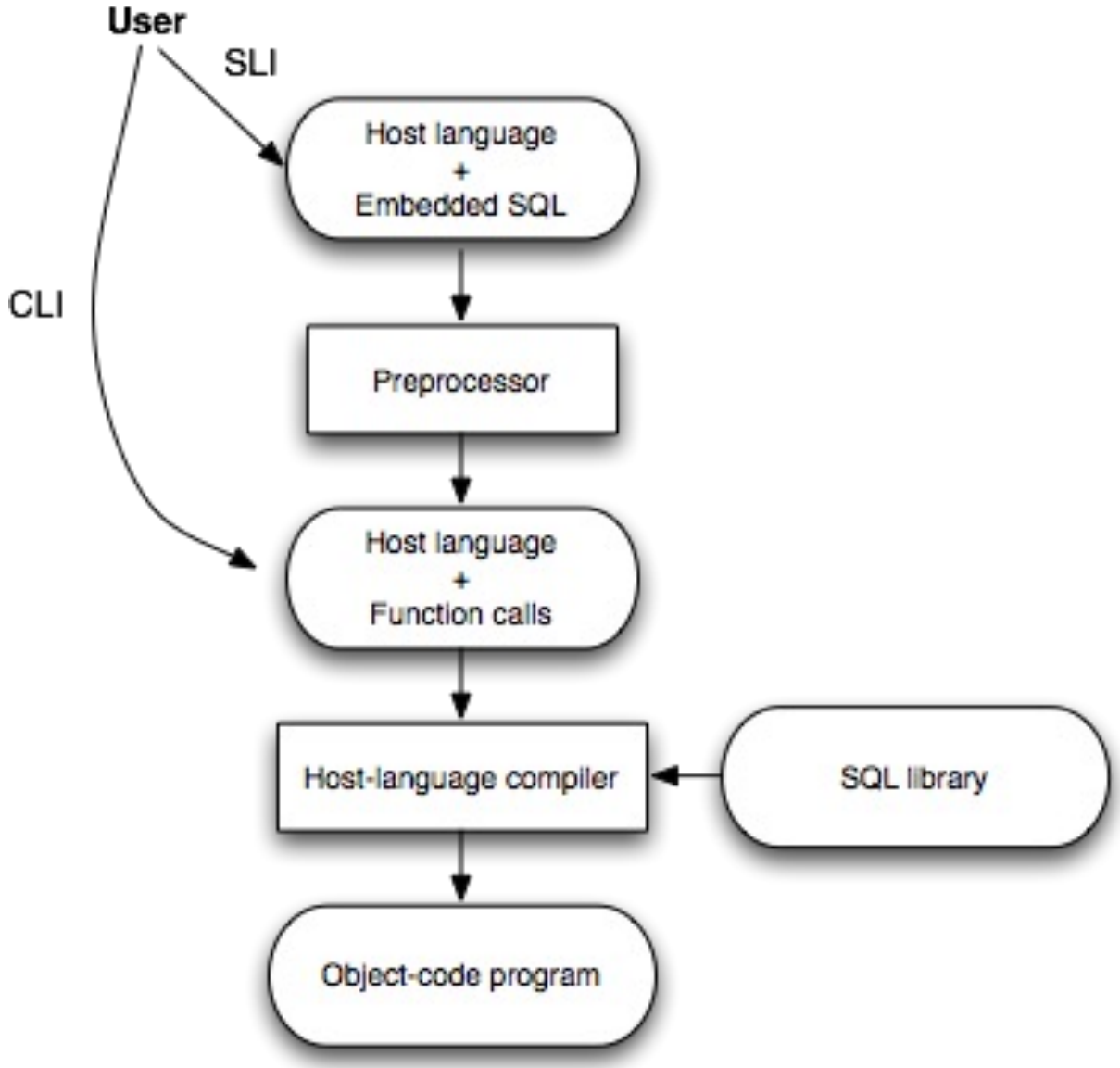
Reference: textbook example 9.7

```
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 */
```



Big picture

figure 9.5, Ulman and Widom



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 library for this. Examples:
  - C has SQL/CLI
  - Java has JDBC
- We'll look at psycopg2 for Python.
  - Can connect a Python program to PostgreSQL, MySQL, SQLite, etc. We'll use it with PostgreSQL of course.

The psycopg2 Library for Python

Demo: A static query

For full details on the classes and methods used, see the documentation:
https://www.psycopg.org/docs/

Aside: where to run psycopg2 code

- Our database server (`dbsrv1`) is configured so that you can only connect to it from that machine.
- So you must run your psycopg2 code on `dbsrv1`.
- This configuration is for security.

If the query isn't known in advance

What if the query depends on something, e.g.,

- The result of a computation
- Input from somewhere

Then we can't write the full query in quotes.

Demo: A dynamic query

SQL Injections

- We just saw an example of an injection.
- The simple approach of building up a query string and passing it to execute is vulnerable to injections.
- Moral of the story: Don't do that!
- Instead, use the second argument of the execute method to complete a query dynamically at run time.

The other approaches to embedding SQL?

Why would we use one of the other approaches instead of psycopg2?

Stored Procedures, e.g., with PL/pgSQL

- Offer efficiency benefits.
- See: https://www.postgresql.org/docs/14/plpgsql-overview.html#PLPGSQL-ADVANTAGES
- But stored procedures are not very standard, so code is not very portable.