SQL: Programming

Introduction to Databases

CompSci 316 Fall 2018
Announcements (Thu., Oct. 11)

- Dean Khary McGhee, Office of Student Conduct, speaks about the Duke Community Standard

- Project milestone #1 due tonight
  - Only one member per team needs to submit
  - Remember members.txt

- Homework 2 Problems 5 & 7 due next Tue.

- Midterm is being graded
  - Grades and sample solution expected to be ready this weekend
Motivation

• Pros and cons of SQL
  • Very high-level, possible to optimize
  • Not intended for general-purpose computation

• Solutions
  • Augment SQL with constructs from general-purpose programming languages
    • E.g.: SQL/PSM
  • Use SQL together with general-purpose programming languages: many possibilities
    • Through an API, e.g., Python psycopg2
    • Embedded SQL, e.g., in C
    • Automatic object-relational mapping, e.g.: Python SQLAlchemy
    • Extending programming languages with SQL-like constructs, e.g.: LINQ
An “impedance mismatch”

- SQL operates on a set of records at a time
- Typical low-level general-purpose programming languages operate on one record at a time
  - Less of an issue for functional programming languages

Solution: cursor

- **Open** (a result table): position the cursor before the first row
- **Get next**: move the cursor to the next row and return that row; raise a flag if there is no such row
- **Close**: clean up and release DBMS resources

- Found in virtually every database language/API
  - With slightly different syntaxes
- Some support more positioning and movement options, modification at the current position, etc.
Augmenting SQL: SQL/PSM

- PSM = Persistent Stored Modules
- CREATE PROCEDURE proc_name(param_decls) local_decls proc_body;
- CREATE FUNCTION func_name(param_decls) RETURNS return_type local_decls func_body;
- CALL proc_name(params);
- Inside procedure body:
  SET variable = CALL func_name(params);
CREATE FUNCTION SetMaxPop(IN newMaxPop FLOAT) RETURNS INT
   -- Enforce newMaxPop; return # rows modified.
BEGIN
   DECLARE rowsUpdated INT DEFAULT 0;
   DECLARE thisPop FLOAT;
   
   -- A cursor to range over all users:
   DECLARE userCursor CURSOR FOR
      SELECT pop FROM User
      FOR UPDATE;
   
   -- Set a flag upon “not found” exception:
   DECLARE noMoreRows INT DEFAULT 0;
   DECLARE CONTINUE HANDLER FOR NOT FOUND
      SET noMoreRows = 1;
   
... (see next slide) ...

RETURN rowsUpdated;
END
-- Fetch the first result row:
OPEN userCursor;
FETCH FROM userCursor INTO thisPop;
-- Loop over all result rows:
WHILE noMoreRows <> 1 DO
    IF thisPop > newMaxPop THEN
        -- Enforce newMaxPop:
        UPDATE User SET pop = newMaxPop
        WHERE CURRENT OF userCursor;
        -- Update count:
        SET rowsUpdated = rowsUpdated + 1;
    END IF;
    -- Fetch the next result row:
    FETCH FROM userCursor INTO thisPop;
END WHILE;
CLOSE userCursor;
Other SQL/PSM features

• Assignment using scalar query results
  • SELECT INTO

• Other loop constructs
  • FOR, REPEAT UNTIL, LOOP

• Flow control
  • GOTO

• Exceptions
  • SIGNAL, RESIGNAL

...

• For more PostgreSQL-specific information, look for “PL/pgSQL” in PostgreSQL documentation
  • Link available from course website (under Help: PostgreSQL Tips)
Working with SQL through an API

• E.g.: Python psycopg2, JDBC, ODBC (C/C++/VB)
  • All based on the SQL/CLI (Call-Level Interface) standard

• The application program sends SQL commands to the DBMS at runtime
• Responses/results are converted to objects in the application program
Example API: Python psycopg2

```python
import psycopg2
conn = psycopg2.connect(dbname='beers')
cur = conn.cursor()

# list all drinkers:
cur.execute('SELECT * FROM Drinker')
for drinker, address in cur:
    print(drinker + ' lives at ' + address)

# print menu for bars whose name contains “a”: 
cur.execute('SELECT * FROM Serves WHERE bar LIKE %s', ('%a%',))
for bar, beer, price in cur:
    print('{} serves {} at ${:,.2f}'.format(bar, beer, price))

cur.close()
conn.close()
```