SQL: Programming

CPS 116

Introduction to Database Systems

Announcements (September 25)

- Homework #2 due this Thursday
  - Submit to Yi—not through Jun's office door
  - Solution available this weekend
- No class this Thursday
- Midterm in class next Thursday (October 4)
  - Open book, open notes
  - Format similar to the sample midterm
    - Solution available this weekend
  - Covers everything up to next Tuesday's lecture
  - Emphasizes materials exercised in homeworks
- Project milestone #1 due in 2½ weeks

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 (SQL/PSM)
  - Use SQL together with general-purpose programming languages (JDBC, embedded SQL, etc.)

Impedance mismatch and a solution

- SQL operates on a set of records at a time
- Typical low-level general-purpose programming languages operates on one record at a time

☞ 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 (analogous to view update), etc.

Augmenting SQL: SQL/PSM

- PSM = Persistent Stored Modules
- CREATE PROCEDURE *proc_name* (*parameter_declarations*)
  local_declarations
  procedure_body;
- CREATE FUNCTION *func_name* (*parameter_declarations*)
  RETURNS *return_type*
  local_declarations
  procedure_body;
- CALL *proc_name* (*parameters*);
- Inside procedure body:
  SET *variable* = CALL *func_name* (*parameters*);

SQL/PSM example

```
CREATE FUNCTION SetMaxGPA(IN newMaxGPA FLOAT)
   RETURNS INT
   -- Enforce newMaxGPA; return number of rows modified.
BEGIN
   DECLARE rowsUpdated INT DEFAULT 0;
   DECLARE thisGPA FLOAT;
   -- A cursor to range over all students:
   DECLARE studentCursor CURSOR FOR
       SELECT GPA FROM Student
   FOR UPDATE;
   -- Set a flag whenever there is a "not found" exception:
   DECLARE noMoreRows INT DEFAULT 0;
   DECLARE CONTINUE HANDLER FOR NOT FOUND
       SET noMoreRows = 1;
   ... (see next slide) ...
   RETURN rowsUpdated;
END
```

SQL/PSM example continued

```
-- Fetch the first result row:
OPEN studentCursor;
FETCH FROM studentCursor INTO thisGPA;
-- Loop over all result rows:
WHILE noMoreRows <> 1 DO
    IF thisGPA > newMaxGPA THEN
        -- Enforce newMaxGPA:
        UPDATE Student SET Student.GPA = newMaxGPA
        WHERE CURRENT OF studentCursor;
        -- Update count:
        SET rowsUpdated = rowsUpdated + 1;
    END IF;
    -- Fetch the next result row:
    FETCH FROM studentCursor INTO thisGPA;
END WHILE;
CLOSE studentCursor;
```

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 DB2-specific information, check out *Developing SQL and External Routines*
  - Available as part of DB2 v9 manual collection, or directly as ftp://ftp.software.ibm.com/ps/products/db2/info/vr9/pdf/letter/en_US/db2z0e90.pdf

Interfacing SQL with another language

- API approach
  - SQL commands are sent to the DBMS at runtime
  - Examples: JDBC, ODBC (for C/C++/VB), Perl DBI
  - These API's are all based on the SQL/CLI (Call-Level Interface) standard
- Embedded SQL approach
  - SQL commands are embedded in application code
  - A precompiler checks these commands at compile-time and converts them into DBMS-specific API calls
  - Examples: embedded SQL for C/C++, SQLJ (for Java)