Real SQL Programming

Persistent Stored Modules (PSM)
MySQL Specifics

SQL in Real Programs

- We have seen only how SQL is used at the generic query interface --- an environment where we sit at a terminal and ask queries of a database.
- Reality is almost always different: conventional programs interacting with SQL.

Options

1. Code in a specialized language is stored in the database itself (e.g., PSM, PL/SQL).
2. SQL statements are embedded in a host language (e.g., C).
3. Connection tools are used to allow a conventional language to access a database (e.g., CLI, JDBC, PHP/DB).

Recap before we go on.

1. We’ve covered chapters 1, 2, and 6.
2. We’re now in Chapter 9.
   - But, we are not going to cover the C-specific aspects.
   - Jump right into 9.4 – PSMs.
   - From there, we will jump to 9.7

Stored Procedures

- PSM, or “persistent stored modules,” allows us to store procedures as database schema elements.
- PSM = a mixture of conventional statements (if, while, etc.) and SQL.
- Lets us do things we cannot do in SQL alone.

Basic PSM Form

CREATE PROCEDURE <name> (  
  <parameter list>  )  
  <optional local declarations>  
  <body>;  

Function alternative:
CREATE FUNCTION <name> (  
  <parameter list>  )  RETURNS <type>
Parameters in PSM

Unlike the usual name-type pairs in languages like C, PSM uses mode-name-type triples, where the **mode** can be:
- **IN** = procedure uses value, does not change value.
- **OUT** = procedure changes, does not use.
- **INOUT** = both.

Example: Stored Procedure

Let’s write a procedure that takes two arguments \( b \) and \( p \), and adds a tuple to \( \text{Sells}(\text{bar, beer, price}) \) that has \( \text{bar} = ‘\text{Joe’s Bar’}, \text{beer} = b, \text{and price} = p. \)
- Used by Joe to add to his menu more easily.

The Procedure

CREATE PROCEDURE JoeMenu (  
    IN b CHAR(20),  
    IN p REAL  
)  
BEGIN  
    INSERT INTO Sells  
    VALUES(‘Joe’s Bar’, b, p);  
END

Invoking Procedures

Use SQL/PSM statement CALL, with the name of the desired procedure and arguments.

Example:

CALL JoeMenu(‘Moosedrool’, 5.00);

Functions used in SQL expressions wherever a value of their return type is appropriate.

Kinds of PSM statements – (1)

- **RETURN** <expression> sets the return value of a function.
  - Unlike C, etc., **RETURN** *does not* terminate function execution.
- **DECLARE** <name> <type> used to declare local variables.
- **BEGIN . . . END** for groups of statements.
  - Separate statements by semicolons.

Kinds of PSM Statements – (2)

- **Assignment statements**:
  - **SET** <variable> = <expression>;
  - Example: **SET** b = ‘Bud’;
- **Statement labels**: give a statement a label by prefixing a name and a colon.
IF Statements

- Simplest form:
  
  IF <condition> THEN
  <statements(s)>
  END IF;

- Add ELSE <statement(s)> if desired, as
  IF . . . THEN . . . ELSE . . . END IF;

- Add additional cases by ELSEIF
  <statements(s)>:
  IF ... THEN ... ELSEIF ... THEN ... ELSEIF ... THEN ... ELSE ... END IF;

Example: IF

- Let’s rate bars by how many customers they have, based on Frequents(drinker,bar).
  - <100 customers: ‘unpopular’.
  - 100-199 customers: ‘average’.
  - >= 200 customers: ‘popular’.
- Function Rate(b) rates bar b.

Example: IF (continued)

CREATE FUNCTION Rate (IN b CHAR(20) )
RETURNS CHAR(10)
DECLARE cust INTEGER;
BEGIN
  SET cust = (SELECT COUNT(*) FROM Frequents
  WHERE bar = b);
  IF cust < 100 THEN RETURN 'unpopular'
  ELSEIF cust < 200 THEN RETURN 'average'
  ELSE RETURN 'popular'
  END IF;
END;

Loops

- Basic form:
  <loop name>: LOOP
  <statements>
  END LOOP;

- Exit from a loop by:
  LEAVE <loop name>

Example: Exiting a Loop

loop1: LOOP
  ... 
  LEAVE loop1;
  ... 
END LOOP;

Other Loop Forms

- WHILE <condition>
  DO <statements>
  END WHILE;

- REPEAT <statements>
  UNTIL <condition>
  END REPEAT;
Queries

- General SELECT-FROM-WHERE queries are not permitted in PSM.
- There are three ways to get the effect of a query:
  1. Queries producing one value can be the expression in an assignment.
  2. Single-row SELECT . . . INTO.
  3. Cursors.

Example: Assignment/Query

- Using local variable p and Sells(bar, beer, price), we can get the price Joe charges for Bud by:
  
  ```
  SET p = (SELECT price FROM Sells
             WHERE bar = 'Joe''s Bar' AND
             beer = 'Bud');
  ```

SELECT . . . INTO

- Another way to get the value of a query that returns one tuple is by placing INTO <variable> after the SELECT clause.
- Example:
  
  ```
  SELECT price INTO p FROM Sells
             WHERE bar = 'Joe''s Bar' AND
             beer = 'Bud';
  ```

Cursors

- A cursor is essentially a tuple-variable that ranges over all tuples in the result of some query.
- Declare a cursor c by:
  
  ```
  DECLARE c CURSOR FOR <query>;
  ```

Opening and Closing Cursors

- To use cursor c, we must issue the command:
  
  ```
  OPEN c;
  ```
  
  The query of c is evaluated, and c is set to point to the first tuple of the result.
- When finished with c, issue command:
  
  ```
  CLOSE c;
  ```

Fetching Tuples From a Cursor

- To get the next tuple from cursor c, issue command:
  
  ```
  FETCH FROM c INTO x1, x2,...,xn ;
  ```
- The x's are a list of variables, one for each component of the tuples referred to by c.
- c is moved automatically to the next tuple.
Breaking Cursor Loops – (1)

- The usual way to use a cursor is to create a loop with a FETCH statement, and do something with each tuple fetched.
- A tricky point is how we get out of the loop when the cursor has no more tuples to deliver.

Breaking Cursor Loops – (2)

- Each SQL operation returns a status, which is a 5-digit character string.
  - For example, 00000 = “Everything OK,” and 02000 = “Failed to find a tuple.”
- In PSM, we can get the value of the status in a variable called SQLSTATE.

Breaking Cursor Loops – (3)

- We may declare a condition, which is a boolean variable that is true if and only if SQLSTATE has a particular value.
- Example: We can declare condition NotFound to represent 02000 by:
  ```sql
  DECLARE NotFound CONDITION FOR SQLSTATE '02000';
  ```

Breaking Cursor Loops – (4)

- The structure of a cursor loop is thus:
  ```sql
  cursorLoop: LOOP 
  ... 
  FETCH c INTO ... ;
  IF NotFound THEN LEAVE cursorLoop;
  END IF;
  ...
  END LOOP;
  ```

Example: Cursor

- Let’s write a procedure that examines Sells(bar, beer, price), and raises by $1 the price of all beers at Joe’s Bar that are under $3.
  - Yes, we could write this as a simple UPDATE, but the details are instructive anyway.

The Needed Declarations

```sql
CREATE PROCEDURE JoeGouge( )
DECLARE theBeer CHAR(20);
DECLARE thePrice REAL;
DECLARE NotFound CONDITION FOR SQLSTATE '02000';
DECLARE c CURSOR FOR
(SELECT beer, price FROM Sells
WHERE bar = 'Joe''s Bar');
```
Exercises Review & Skills

❖ Carry out the tasks in
   ❖ Exercise 9.4.2

Skipping from 9.4 to 9.7

❖ So far this term, we’ve interacted with SQL by direct interaction through the MySQL console.
❖ In “real world environments,” SQL calls are made through CLI (command-line interfaces) using system library bindings.
❖ PHP? Why? Because there are more arguments against all the other languages.
❖ PHP gets noticed on your resume’