PREV
2.9. Calling Stored Programs from Stored Program

NEXT
2.11. Stored Functions

Putting It All Together

In Example 2-7 we show a stored procedure that uses all the features of the stored program language we have covered so far in this tutorial.

Example 2-7. A more complex stored procedure

```
1  CREATE PROCEDURE putting_it_all_together(in_department_id INT)
2      MODIFIES SQL DATA
3  BEGIN
4      DECLARE l_employee_id INT;
5      DECLARE l_salary      NUMERIC(8,2);
6     DECLARE l_department_id INT;
7     DECLARE l_new_salary  NUMERIC(8,2);
8     DECLARE done          INT DEFAULT 0;
9
10    DECLARE cur1 CURSOR FOR
11            SELECT employee_id, salary, department_id
12              FROM employees
13             WHERE department_id=in_department_id;
14
15
16          DECLARE CONTINUE HANDLER FOR NOT FOUND SET done=1;
17
18    CREATE TEMPORARY TABLE IF NOT EXISTS emp_raises
19      (employee_id INT, department_id INT, new_salary NUMERIC(
20
21    OPEN cur1;
22    emp_loop: LOOP
23
24      FETCH cur1 INTO l_employee_id, l_salary, l_department_id
25
26      IF done=1 THEN       /* No more rows*/
27         LEAVE emp_loop;
28      END IF;
29
30      CALL new_salary(l_employee_id,l_new_salary); /*get new s
31
32      IF (l_new_salary<>l_salary) THEN             /*Salary ch
33
34         UPDATE employees
35            SET salary=l_new_salary
36          WHERE employee_id=l_employee_id;
37         /* Keep track of changed salaries*/
38         INSERT INTO emp_raises (employee_id,department_id,new
39           VALUES (l_employee_id,l_department_id,l_new_salary);
40      END IF;
41
42    END LOOP emp_loop;
43    CLOSE cur1;
44    /* Print out the changed salaries*/
45    SELECT employee_id,department_id,new_salary from emp_raise
46     ORDER BY employee_id;
47 END;
```

This is the most complex procedure we have written so far, so let's go through it line by line:

Line(s)	Explanation
1	Create the procedure. It takes a single parameter—`in_department_id`. Since we did not specify the `OUT` or `INOUT` mode, the parameter is for input only (that is, the calling program cannot read any changes to the parameter made within the procedure).
4-8	Declare local variables for use within the procedure. The final parameter, `done`, is given an initial value of 0.
10-13	Create a cursor to retrieve rows from the `employees` table. Only employees from the department passed in as a parameter to the procedure will be retrieved.
16	Create an error handler to deal with "not found" conditions, so that the program will not terminate with an error after the last row is fetched from the cursor. The handler specifies the `CONTINUE` clause, so the program execution will continue after the "not found" error is raised. The handler also specifies that the variable `done` will be set to 1 when this occurs.
18	Create a temporary table to hold a list of rows affected by this procedure. This table, as well as any other temporary tables created in this session, will be dropped automatically when the session terminates.
21	Open our cursor to prepare it to return rows.
22	Create the loop that will execute once for each row returned by the stored procedure. The loop terminates on line 42.
24	Fetch a new row from the cursor into the local variables that were declared earlier in the procedure.
26-28	Declare an `IF` condition that will execute the `LEAVE` statement if the variable `done` is set to 1 (accomplished through the "not found" handler, which means that all rows were fetched).
30	Call the `new_salary` procedure to calculate the employee's new salary. It takes as its arguments the `employee_id` and an `OUT` variable to accept the new salary (`l_new_salary`).
32	Compare the new salary calculated by the procedure called on line 30 with the existing salary returned by the cursor defined on line 10. If they are different, execute the block of code between lines 32 and 40.
34-36	Update the employee salary to the new salary as returned by the `new_salary` procedure.
38 and 39	Insert a row into our temporary table (defined on line 21) to record the salary adjustment.
43	After all of the rows have been processed, close the cursor.

Line(s)	Explanation
45	Issue an unbounded SELECT (e.g., one without a WHERE clause) against the temporary table, retrieving the list of employees whose salaries have been updated. Because the SELECT statement is not associated with a cursor or an INTO clause, the rows retrieved will be returned as a result set to the calling program.
47	Terminate the stored procedure.

When this stored procedure is executed from the MySQL command line with the parameter of `department_id` set to 18, a list of updated salaries is printed as shown in Example 2-8.

Example 2-8. Output from the "putting it all together" example

```
mysql> CALL putting_it_all_together(18) //
+-------------+---------------+------------+
| employee_id | department_id | new_salary |
+-------------+---------------+------------+
|         396 |            18 |   75560.00 |
|         990 |            18 |  118347.00 |
+-------------+---------------+------------+
2 rows in set (0.23 sec)

Query OK, 0 rows affected (0.23 sec)
```