Table of Contents

This is optional material; it might be of interest to people who have done other types of programming.

1. Creating a stored procedure

You can create a named procedure and store it in the database using the same technique as described for functions. You can either create this at the command line or put it in a script file.

There are a few differences between functions and procedures

- you can do insert, updates, and deletes in a procedure
- you cannot use a procedure in an SQL statement because a procedure does not return a value through its name

2. Executing a Stored Procedure

Assume that we created a procedure named proc_01. Use the call command to execute the procedure from the mysql prompt.

```
Call proc_001()#
```

3. Using DML Statements in Procedures

The only procedures I want to show you are a few procedures that do DML statements. One use of procedures is to encapsulate program logic.

Assume we had this table.

```
Create table a_testbed.pets (
   pet_id    int primary key
,  pet_name  varchar(25) not null
,  pet_price numeric(6,2) not null
,  pet_added_date date not null
)
```

Demo 01: The following procedure has three arguments, which are used to get values to insert into the table. The pet_added_date gets the current date.

```
drop procedure if exists a_testbed.insert_pet#

create procedure a_testbed.insert_pet (
     in_pet_id      int
    ,in_pet_name   varchar(25)
    ,in_pet_price  numeric(6,2)
)
begin
   insert into a_testbed.pets (
               pet_id,   pet_name,    pet_price,  pet_added_date)
    values( in_pet_id, in_pet_name, in_pet_price,  curdate()
);
end;
#
```

Demo 02: Use the procedure to add some rows.

```
Call a_testbed.insert_pet( 101, 'Mittens',   50)#
Call a_testbed.insert_pet( 105, 'Spot',      45.28)#
Call a_testbed.insert_pet( 108, 'Curley',  5000)#

Select * from a_testbed.pets#
+--------+----------+-----------+----------------+
| pet_id | pet_name | pet_price | pet_added_date |
+--------+----------+-----------+----------------+
|    101 | Mittens  |     50.00 | 2011-04-27     |
|    105 | Spot     |     45.28 | 2011-04-27     |
|    108 | Curley   |   5000.00 | 2011-04-27     |
+--------+----------+-----------+----------------+
```

The following gives an error message; the procedure expects 3 arguments.

```
Call a_testbed.insert_pet( 120, 'Mittens')#
ERROR 1318 (42000): Incorrect number of arguments for PROCEDURE
a_testbed.insert_pet; expected 3, got 2
```

This simple example does not look like this gives us much- but we could add additional logic inside the procedure to do additional tasks- such as more sophisticated validation of the values or perhaps adding data to more than one table. The procedure could log entries into an audit table that records all of the changes of the price attribute in the pets table and the account that was running the procedure. The procedure could run only during normal working hours. An application could be set up that provides the values for the insert via bind variables that get their data from an on-screen form and passes them to this procedure.

Demo 03: Doing an update a procedure

```
drop procedure if exists a_testbed.change_pet_price #

Create procedure a_testbed.change_pet_price (
     p_pet_ID int, p_price_change numeric(6,2))
begin
   update   a_testbed.pets
      set   pet_price = pet_price + p_price_change
    where   pet_id = p_pet_id;
end;
#
```

Demo 04: Doing an update– set the user variables, then execute the procedure

```
Set @pet_id = 105#
Set @pIncrease = 100#

Call a_testbed.change_pet_price(@pet_id, @pIncrease)#

Select *
from a_testbed.pets#
+--------+----------+-----------+----------------+
| pet_id | pet_name | pet_price | pet_added_date |
+--------+----------+-----------+----------------+
|    101 | Mittens  |     50.00 | 2011-04-27     |
|    105 | Spot     |    145.28 | 2011-04-27     |
|    108 | Curley   |   5000.00 | 2011-04-27     |
+--------+----------+-----------+----------------+
```