------Creating Stored Procedures in MySQL------

--Make sure you have version 5 of MySQL:

SELECT VERSION();

+-----------+

| VERSION() |

+-----------+

| 5.0.15-nt |

+-----------+

1 row in set (0.00 sec)

--First pick a database to use (a procedure, like a table, is associated with

--a single database.) For these examples, I will use a database that is populated

--with the tables from HW 2:

USE ozaidan\_hw2;

--Next, change the delimiter, because we will use the semicolon WITHIN the

--procedure declarations, and therefore it cannot be the delimiter anymore:

DELIMITER //

--OK, let's get started. Creating procedures is straightforward:

CREATE PROCEDURE myFirstProc()

SELECT 'Hello World!' AS Output;

//

Query OK, 0 rows affected (0.00 sec)

--Whenever you create a procedure (successfully) you should get a 'Query OK' message.

--Calling a procedure is also straightforward:

CALL myFirstProc() //

+--------------+

| Output |

+--------------+

| Hello World! |

+--------------+

1 row in set (0.00 sec)

--By the way, procedure names are NOT case sensitive:

CALL myfirstproc() //

+--------------+

| Output |

+--------------+

| Hello World! |

+--------------+

1 row in set (0.00 sec)

--Another example:

CREATE PROCEDURE ListStudents()

SELECT \*

FROM Student;

//

CALL ListStudents() //

+-------+----------+---------+------+------+-------+---------+-----------+

| StuID | LName | Fname | Age | Sex | Major | Advisor | city\_code |

+-------+----------+---------+------+------+-------+---------+-----------+

| 1001 | Smith | Linda | 18 | F | 600 | 1121 | BAL |

| 1002 | Kim | Tracy | 19 | F | 600 | 7712 | HKG |

.

.

.

| 1034 | Epp | Eric | 18 | M | 50 | 5718 | BOS |

| 1035 | Schmidt | Sarah | 26 | F | 50 | 5718 | WAS |

+-------+----------+---------+------+------+-------+---------+-----------+

34 rows in set (0.00 sec)

--Say we only want student ID's and names. To update a procedure, we must

--first DROP it:

DROP PROCEDURE IF EXISTS ListStudents //

Query OK, 0 rows affected (0.00 sec)

--Again, whenever you drop a procedure, you should get a 'Query OK' message.

--From now on, we will always use "DROP PROCEDURE IF EXISTS procName" as

--a standard practice before declaring procedures:

DROP PROCEDURE IF EXISTS ListStudents //

CREATE PROCEDURE ListStudents()

SELECT StuID, LName, FName

FROM Student;

//

CALL ListStudents() //

+-------+----------+---------+

| StuID | LName | FName |

+-------+----------+---------+

| 1001 | Smith | Linda |

| 1002 | Kim | Tracy |

.

.

.

| 1034 | Epp | Eric |

| 1035 | Schmidt | Sarah |

+-------+----------+---------+

34 rows in set (0.00 sec)

--OK, let's use some parameters:

DROP PROCEDURE IF EXISTS sayHello //

CREATE PROCEDURE sayHello(IN name VARCHAR(20))

SELECT CONCAT('Hello ', name, '!') AS Greeting;

//

--The 'IN' keyword tells MySQL that is should be expecting an input value for

--the parameter......hunh? Why would a parameter NOT have an input value? You will

--see in a little bit. First, let's see if sayHello works:

CALL sayHello('Omar') //

+-------------+

| Greeting |

+-------------+

| Hello Omar! |

+-------------+

1 row in set (0.00 sec)

--Another example:

DROP PROCEDURE IF EXISTS saySomething //

CREATE PROCEDURE saySomething(IN phrase VARCHAR(20), IN name VARCHAR(20))

SELECT CONCAT(phrase, ' ', name, '!') AS Output;

//

CALL saySomething('Go','Blue Jays') //

CALL saySomething('Do','my homework') //

+---------------+

| Output |

+---------------+

| Go Blue Jays! |

+---------------+

1 row in set (0.00 sec)

+-----------------+

| Output |

+-----------------+

| Do my homework! |

+-----------------+

1 row in set (0.00 sec)

--and another one:

DROP PROCEDURE IF EXISTS FindStudent //

CREATE PROCEDURE FindStudent(IN id INT)

SELECT StuID, CONCAT(FName, ' ', LName) AS 'Student Name'

FROM Student

WHERE StuID = id;

//

CALL FindStudent(1001) //

+-------+--------------+

| StuID | Student Name |

+-------+--------------+

| 1001 | Linda Smith |

+-------+--------------+

1 row in set (0.00 sec)

--and yet another:

DROP PROCEDURE IF EXISTS calculate //

CREATE PROCEDURE calculate(IN x INT, IN y INT, OUT sum INT, OUT product INT)

SET sum = x + y;

SET product = x \* y;

//

ERROR 1064 (42000): You have an error in your SQL syntax; check the manual ...

--Well, that wasn't good. The reason is, we must use BEGIN/END if we have

--a compound statement:

DROP PROCEDURE IF EXISTS calculate //

CREATE PROCEDURE calculate(IN x INT, IN y INT, OUT sum INT, OUT product INT)

BEGIN

SET sum = x + y;

SET product = x \* y;

END;

//

--Did you notice the 'OUT' keyword for sum and product? This tells MySQL that those

--two parameters are not 'input' parameters but are 'output' parameters instead.

--Now, when calling the procedure, we need to provide four parameters: two input

--values, and two MySQL \*variables\* where the results will be stored:

CALL calculate(4,5,@s,@p) //

Query OK, 0 rows affected (0.00 sec)

--Here, @s and @p are MySQL variables. Notice that they start with @, although

--procedure \*parameters\* do not start with @

SELECT @s //

SELECT @p //

+------+

| @s |

+------+

| 9 |

+------+

1 row in set (0.00 sec)

+------+

| @p |

+------+

| 20 |

+------+

1 row in set (0.00 sec)

--Note: you can also have INOUT parameters, which serve as both input and output

--parameters.

--OK, let's do some interesting stuff. First off, flow control:

DROP PROCEDURE IF EXISTS mySign //

CREATE PROCEDURE mySign(IN x INT)

BEGIN

IF x > 0 THEN

SELECT x AS Number, '+' AS Sign;

ELSEIF x < 0 THEN

SELECT x AS Number, '-' AS Sign;

ELSE

SELECT x AS Number, 'Zero' AS Sign;

END IF;

END;

//

CALL mySign(2) //

CALL mySign(-5) //

CALL mySign(0) //

+--------+------+

| Number | Sign |

+--------+------+

| 2 | + |

+--------+------+

1 row in set (0.00 sec)

+--------+------+

| Number | Sign |

+--------+------+

| -5 | - |

+--------+------+

1 row in set (0.00 sec)

+--------+------+

| Number | Sign |

+--------+------+

| 0 | Zero |

+--------+------+

1 row in set (0.00 sec)

--Before we get any further, let's introduce variables:

DROP PROCEDURE IF EXISTS mySign //

CREATE PROCEDURE mySign(IN x INT)

BEGIN

DECLARE result VARCHAR(20);

IF x > 0 THEN

SET result = '+';

ELSEIF x < 0 THEN

SET result = '-';

ELSE

SET result = 'Zero';

END IF;

SELECT x AS Number, result AS Sign;

END;

//

CALL mySign(2) //

CALL mySign(-5) //

CALL mySign(0) //

+--------+------+

| Number | Sign |

+--------+------+

| 2 | + |

+--------+------+

1 row in set (0.00 sec)

+--------+------+

| Number | Sign |

+--------+------+

| -5 | - |

+--------+------+

1 row in set (0.00 sec)

+--------+------+

| Number | Sign |

+--------+------+

| 0 | Zero |

+--------+------+

1 row in set (0.00 sec)

--Using CASE:

DROP PROCEDURE IF EXISTS digitName //

CREATE PROCEDURE digitName(IN x INT)

BEGIN

DECLARE result VARCHAR(20);

CASE x

WHEN 0 THEN SET result = 'Zero';

WHEN 1 THEN SET result = 'One';

WHEN 2 THEN SET result = 'Two';

WHEN 3 THEN SET result = 'Three';

WHEN 4 THEN SET result = 'Four';

WHEN 5 THEN SET result = 'Five';

WHEN 6 THEN SET result = 'Six';

WHEN 7 THEN SET result = 'Seven';

WHEN 8 THEN SET result = 'Eight';

WHEN 9 THEN SET result = 'Nine';

ELSE SET result = 'Not a digit';

END CASE;

SELECT x AS Digit, result AS Name;

END;

//

CALL digitName(0) //

CALL digitName(4) //

CALL digitName(100) //

+-------+------+

| Digit | Name |

+-------+------+

| 0 | Zero |

+-------+------+

1 row in set (0.00 sec)

+-------+------+

| Digit | Name |

+-------+------+

| 4 | Four |

+-------+------+

1 row in set (0.00 sec)

+-------+-------------+

| Digit | Name |

+-------+-------------+

| 100 | Not a digit |

+-------+-------------+

1 row in set (0.00 sec)

--As you'd expect, we have loops. For example, WHILE loops:

DROP PROCEDURE IF EXISTS fact //

CREATE PROCEDURE fact(IN x INT)

BEGIN

DECLARE result INT;

DECLARE i INT;

SET result = 1;

SET i = 1;

WHILE i <= x DO

SET result = result \* i;

SET i = i + 1;

END WHILE;

SELECT x AS Number, result as Factorial;

END;

//

CALL fact(1) //

CALL fact(2) //

CALL fact(4) //

CALL fact(0) //

+--------+-----------+

| Number | Factorial |

+--------+-----------+

| 1 | 1 |

+--------+-----------+

1 row in set (0.00 sec)

+--------+-----------+

| Number | Factorial |

+--------+-----------+

| 2 | 2 |

+--------+-----------+

1 row in set (0.00 sec)

+--------+-----------+

| Number | Factorial |

+--------+-----------+

| 4 | 24 |

+--------+-----------+

1 row in set (0.01 sec)

+--------+-----------+

| Number | Factorial |

+--------+-----------+

| 0 | 1 |

+--------+-----------+

1 row in set (0.00 sec)

--There is also REPEAT/UNTIL loops:

DROP PROCEDURE IF EXISTS fact //

CREATE PROCEDURE fact(IN x INT)

BEGIN

DECLARE result INT DEFAULT 1; /\* notice you can declare a variable\*/

DECLARE i INT DEFAULT 1; /\* and give it a value in one line \*/

REPEAT

SET result = result \* i;

SET i = i + 1;

UNTIL i > x

END REPEAT;

SELECT x AS Number, result as Factorial;

END;

//

CALL fact(1) //

CALL fact(2) //

CALL fact(4) //

CALL fact(0) //

+--------+-----------+

| Number | Factorial |

+--------+-----------+

| 1 | 1 |

+--------+-----------+

1 row in set (0.00 sec)

+--------+-----------+

| Number | Factorial |

+--------+-----------+

| 2 | 2 |

+--------+-----------+

1 row in set (0.00 sec)

+--------+-----------+

| Number | Factorial |

+--------+-----------+

| 4 | 24 |

+--------+-----------+

1 row in set (0.00 sec)

+--------+-----------+

| Number | Factorial |

+--------+-----------+

| 0 | 1 |

+--------+-----------+

1 row in set (0.00 sec)

--OK, do you remember this?

/\*

CREATE PROCEDURE FindStudent(IN id INT)

SELECT StuID, CONCAT(FName, ' ', LName) AS 'Student Name'

FROM Student

WHERE StuID = id;

//

\*/

--What if we only want to extract the name without printing it out?

--Obviously, we need some OUT parameters. Still, how do you extract

--information into those OUT parameters?

--

--Answer: something called a CURSOR:

DROP PROCEDURE IF EXISTS FindName //

CREATE PROCEDURE FindName(IN id INT, OUT fn VARCHAR(20), OUT ln VARCHAR(20))

BEGIN

DECLARE cur CURSOR FOR

SELECT FName, LName

FROM Student

WHERE StuID = id;

OPEN cur;

FETCH cur INTO fn, ln;

CLOSE cur;

END;

//

CALL FindName(1001,@f,@l) //

Query OK, 0 rows affected (0.00 sec)

--Remember that @f and @l are MySQL variables:

SELECT @f //

SELECT @l //

+-------+

| @f |

+-------+

| Linda |

+-------+

1 row in set (0.00 sec)

+-------+

| @l |

+-------+

| Smith |

+-------+

1 row in set (0.00 sec)

--What if we give an invalid student ID?

CALL FindName(0000,@f,@l) //

ERROR 1329 (02000): No data to FETCH

--MySQL complains, as expected. It would be nice to handle this more elegantly, however.

--We need an error HANDLER. Let's modify FindName:

DROP PROCEDURE IF EXISTS FindName //

CREATE PROCEDURE FindName(IN id INT, OUT fn VARCHAR(20), OUT ln VARCHAR(20))

BEGIN

DECLARE cur CURSOR FOR

SELECT FName, LName

FROM Student

WHERE StuID = id;

DECLARE EXIT HANDLER FOR NOT FOUND

SELECT 'Sorry; this ID was not found' AS 'Error Message';

OPEN cur;

FETCH cur INTO fn, ln;

CLOSE cur;

END;

//

CALL FindName(0000,@f,@l) //

+------------------------------+

| Error Message |

+------------------------------+

| Sorry; this ID was not found |

+------------------------------+

1 row in set (0.00 sec)

--Another use for handlers: multiple FETCH calls using a CONTINUE handler.

--

--In this case, we use a CONTINUE handler that, instead of exiting the procedure

--upon encountering a NOT FOUND error, simply sets a variable done = 1.

--

--Why would we do that? And how does that help us carry out multiple FETCH calls?

--

--Take a look at this procedure, which traverses all the entries of a table to

--find the maximum and minimum age:

DROP PROCEDURE IF EXISTS MaxMinAge //

CREATE PROCEDURE MaxMinAge(OUT maxAge INT, OUT minAge INT)

BEGIN

DECLARE currAge,maxSoFar,minSoFar,done INT;

DECLARE cur CURSOR FOR

SELECT Age

FROM Student;

DECLARE CONTINUE HANDLER FOR NOT FOUND

SET done = 1;

SET maxSoFar = 0;

SET minSoFar = 1000;

SET done = 0;

OPEN cur;

WHILE done = 0 DO

FETCH cur INTO currAge;

IF currAge > maxSoFar THEN

SET maxSoFar = currAge;

END IF;

IF currAge < minSoFar THEN

SET minSoFar = currAge;

END IF;

END WHILE;

CLOSE cur;

SET maxAge = maxSoFar;

SET minAge = minSoFar;

END;

//

CALL MaxMinAge(@max,@min) //

Query OK, 0 rows affected (0.00 sec)

SELECT @max //

SELECT @min //

+------+

| @max |

+------+

| 27 |

+------+

1 row in set (0.00 sec)

+------+

| @min |

+------+

| 16 |

+------+

1 row in set (0.00 sec)

--In summary, stored procedures in MySQL look like this:

DROP PROCEDURE IF EXISTS procName //

CREATE PROCEDURE procName(parameter list)

BEGIN

/\* variable declarations \*/

/\* CURSOR definitions \*/

/\* declaring handlers \*/

/\* procedure body...whatever you want it to do \*/

END;

//

--In more detail:

DROP PROCEDURE IF EXISTS procName //

CREATE PROCEDURE procName(IN/OUT/INOUT parName parType, ...)

BEGIN

/\* variable declarations \*/

DECLARE varName,... varType;

/\* e.g. DECLARE myName VARCHAR(20); DECLARE x,y,z INT; \*/

DECLARE varName varType DEFAULT value;

/\* e.g. DECLARE x INT DEFAULT 0; \*/

/\* CURSOR definitions \*/

DECLARE curName CURSOR FOR

SELECT ...

/\* e.g. DECLARE cur1 CURSOR FOR

SELECT FName, LName

FROM Student; \*/

/\* declaring handlers \*/

DECLARE EXIT/CONTINUE HANDLER FOR errorType/errorNumber

... action ...

/\* e.g. DECLARE EXIT HANDLER FOR NOT FOUND

SELECT 'Sorry; this ID was not found' AS 'Error Message'; \*/

/\* e.g. DECLARE CONTINUE HANDLER FOR NOT FOUND

SET done = 1; \*/

/\* procedure body...whatever you want it to do \*/

/\* IF statement \*/

IF cond1 THEN

action1

ELSEIF cond2 THEN

action2

ELSEIF cond3 THEN

action3

ELSE

elseaction

END IF;

/\* e.g. IF x > 0 THEN

SET result = '+';

ELSEIF x < 0 THEN

SET result = '-';

ELSE

SET result = 'Zero';

END IF; \*/

/\* CASE statement \*/

CASE varName

WHEN val1 THEN action1

WHEN val2 THEN action2

ELSE elseaction

END CASE;

/\* e.g. CASE position

WHEN 1 THEN SET result = 'Gold Medal';

WHEN 2 THEN SET result = 'Silver Medal';

WHEN 3 THEN SET result = 'Bronze Medal';

ELSE SET result = 'No Medal!';

END CASE; \*/

/\* WHILE loop \*/

WHILE cond DO

action1

action2

...

END WHILE

/\* e.g. WHILE i < 5 DO

SET result = result + i;

SET i = i + 1;

END WHILE; \*/

/\* REPEAT/UNTIL loop \*/

REPEAT

action1

action2

...

UNTIL cond

END REPEAT;

/\* e.g. REPEAT

SET result = result + i;

SET i = i + 1;

UNTIL i >= 5

END REPEAT; \*/

/\* using a CURSOR \*/

OPEN curName;

.

.

FETCH curName INTO var1, var2, ...;

.

.

CLOSE curName;

/\* e.g. Assume cur1 has id's, first names, and last names

let's find the name of the student whose StuID is x:

OPEN cur1;

SET found = 0;

WHILE found = 0 DO

FETCH cur1 INTO nextID, nextFName, nextLName;

IF nextID = x THEN

SET result = CONCAT(nextFName, ' ', nextLName);

SET found = 1;

END IF;

END WHILE;

CLOSE cur1; \*/

END;

//